DIETI. UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

Tesi di Dottorato di Ricerca in Ingegneria Informatica ed Automatica
Coordinatore: Prof. Francesco Garofalo
Aprile 2013

Finding Unexplained Activities in Time-Stamped Observation Data

by

Fabio Persia

Thesis Supervisors: **Prof.Antonio Picariello**

Eng. Vincenzo Moscato

Prof. V.S. Subrahmanian

Submitted to the Faculty of Engineering, University of Naples Federico II, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Printed in Italy.
Napoli, April 2013.

To the best Gift I've ever received, Daniela!

Acknowledgements

This work would certainly not have been possible without the support of many people. First of all, I'd like to acknowledge my Ph.D. thesis' advisors, Prof. Antonio Picariello and Eng. Vincenzo Moscato, for having guided me with great competence and enthusiasm during these years and for having given me the unique opportunity to carry out my doctoral studies at the University of Maryland. So, I'd like to thank Prof. V.S. Subrahmanian at University of Maryland, my Ph.D. thesis' co-advisor, for having given me the chance to spend several months at his laboratory and start the challenging projects around which the work presented in this thesis is built. I'd like to acknowledge Prof. Francesco Garofalo, chair of Naples' Ph.D. School in Computer Science and Engineering.

I'd also like to thank with all my heart Daniela, who is the girl that I want to spend the rest of my life with! Then, I'd like to thank my father, for having guided me from the Heaven and my mother, for her moral and financial support. Finally, I'd like to acknowledge myself for having had the willingness to pursue this goal with great sacrifice.

Abstract

The activity recognition is a very big challenge for the entire research community. Thus, there are already numerous techniques able to find occurrences of activities in time-stamped observation data (e.g., a video, a sequence of transactions at a website, etc.) with each occurrence having an associated probability.

However, all these techniques rely on models encoding a priori knowledge of either normal or malicious behavior. They cannot deal with events such as "zero day" attacks that have never been seen before. In practice, all these methods are incapable of quantifying how well available models explain a sequence of events observed in an observation stream.

By the way, the goal of this thesis is different: in order to address the issue listed above, we want to find the subsequences of the observation data, called *unexplained sequences*, that known models are not able to "explain" with a certain confidence.

Thus, we start with a known set $\mathcal{A}$ of activities (both innocuous and dangerous) that we wish to monitor and we wish to identify "unexplained" subsequences in an observation sequence that are poorly explained (e.g., because they may contain occurrences of activities that have never been seen or anticipated before, i.e. they are not in $\mathcal{A}$).

We formally define the probability that a sequence of observations is unexplained (*totally* or *partially*) w.r.t. $\mathcal{A}$. We develop efficient algorithms to identify the top-k *Totally* and *Partially Unexplained Activities* w.r.t. $\mathcal{A}$. These algorithms leverage theorems that enable us to speed up the search for totally/partially unexplained activities. We describe experiments using real-world video and cyber security datasets showing that our approach works well in practice in terms of both running time and accuracy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The "Unexplained" Activity Problem

Identifying unexpected activities is an important problem in a wide variety of applications such as video surveillance, cyber security, fault detection in safety critical systems, and fraud detection.

For instance, airport baggage areas are continuously monitored for suspicious activities by video surveillance. In crime-ridden neighborhoods, police often monitor streets and parking lots using video surveillance. In Israel, highways are monitored for suspicious activities by a central authority. However, all these applications search for *known* activities—activities that have been identified in advance as being either innocuous or dangerous. For instance, in the highway application, security officers may look both for normal behavior (e.g. driving along the highway in a certain speed range unless traffic is slow) as well as "suspicious" behavior (e.g. stopping the car near a bridge, taking a package out and leaving it on the side of the road before driving away).

In cyber security, intrusion detection can monitor network traffic for suspicious behavior and trigger security alerts. Alert correlation methods aggregate alerts into multi-step attack scenarios. However, both techniques rely on models encoding a priori knowledge of either normal or malicious behavior. They cannot deal with events such as "zero day" attacks that have never been seen before. In practice, all these methods are incapable of quantifying how well available models explain a sequence of events observed in an observation stream.

1.2 Thesis goal

In order to address the issues described in section 1.1 , we designed and implemented a framework (the *Unexplained Sequence Detector*), able to discover the subsequences of the observation stream not sufficiently explained by well-known activity models.

Figure 1.1 shows how our framework would work in practice. We start with a set of activity models $\mathcal{A}$ for both "good" and "bad" activities. Good activities are activities that are considered appropriate (e.g., certain permitted behaviors in an airport secure baggage zone) while bad activities are ones known to be inappropriate (e.g., a baggage handler opening a suitcase, taking items out, and putting them in a different bag). Techniques already exist to find occurrences of activities in time-stamped observation data (e.g., a video, a sequence of transactions at a website, etc.) with each occurrence having an associated probability. In this thesis, our goal is to find an *unexplained sequence detector*, i.e. to identify subsequences of the observation data, called *unexplained sequences*, that known models are not able to "explain" with a certain confidence. In other words, what is happening in unexplained sequences is not well captured by the available activity models in $\mathcal{A}$. Once such subsequences have been identified, they can be further analyzed, e.g.,

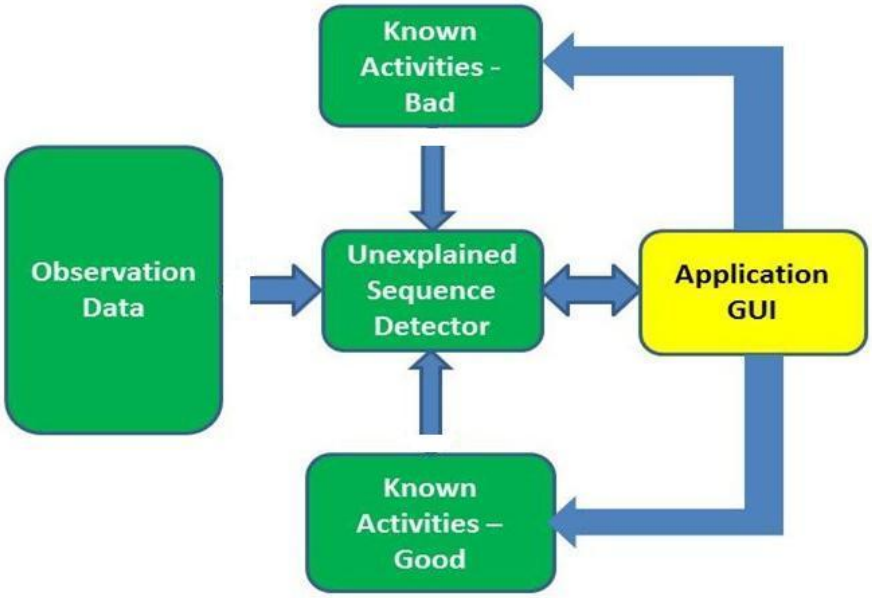


Figure 1.1: Overall working of unexplained sequences

to learn new activity models from them. Or, as shown in Figure 1.1, each unexplained sequence can be shown to a domain expert (e.g., airport security or cyber security expert) who can then add these observed sequences or generalizations thereof to the currently known list of good or bad activities.

Unexplained sequences allow an application to identify activities never seen or imagined before by experts, and to add them to an increasing body of such knowledge. For instance, a new type of terrorist attack at an airport or a zero-day attack on a computer system, may involve sequences of actions (observations) not seen before—and hence not captured by past activity models (i.e., those in $\mathcal{A}$). In this thesis, we primarily focus on the unexplained sequence detector component of Figure 1.1.

We achieve this via a possible-worlds based model and define the probability that a sequence of observations is *totally* (or *partially*) unexplained. Users can then look for all observation sequences that are totally (or partially) unexplained with a probability exceeding a threshold that they specify. We show important properties of our mathematical model that can be leveraged to speed up the search for unexplained activities. We define algorithms to find top-k totally and partially unexplained activities. We develop a prototype implementation and report on experiments using two video data sets and a cyber security dataset showing that the algorithms work well in practice, both from an efficiency perspective and an accuracy perspective.

1.3 Thesis outline

The paper starts (Chapter 2) with an overview of related work. Chapter 3 provides basic definitions of stochastic activities slightly extending [1] (section 3.1), defines the probability that a sequence is totally (or partially) unexplained, the problem of finding the top-k (totally or partially) unexplained activities and classes (section 3.2) and finally derives theorems that enable fast search for totally and partially unexplained activities (subsection 3.2.1). Chapter 4 presents algorithms for solving the problems introduced in Chapter 3. Chapter 5 describes our prototype implementations developed in the video surveillance and cyber security domains and the related experiments. Chapter 6 presents the conclusions of the thesis and the possible future works.

Chapter 2

Related Work

We are not aware of domain-independent prior work on discovering unexplained activities. However, specific work in the domains of video and cyber-security have focused on anomalous activity detection.

2.1 Video Analysis

2.1.1 A Priori Definitions.

Several researchers have studied how to search for specifically defined patterns of normal/abnormal activities [2].

The most relevant approaches following this philosophy are:

- DBN (Dynamic Bayesian Networks)
- HMM (Hidden Markov Models)

2.1.1.1 DBN (Dynamic Bayesian Networks)

Bayesian Networks have been used in many researches; for instance, Buxton used Bayesian Belief Networks for the video interpretation in an application of traffic surveillance [3]. Huang [4] used them for the video interpretation of an airport scenery, Nevatia [5] to single out the interactions in a group of people. Intille e Bobick [6] used Bayesian Networks to single out numerous activities during a football match. Notwithstanding this, no information is supplied on how the learning of the net parameters occurs.

Besides the learning problem, the BN have a further defect. They are not suited to model temporal relationships. As a matter of fact, the time during which the net building takes place is to be explicitly pointed out. Particular dynamic Bayesian Nets, the recurrent Bayesian Networks, have been used for the singling out of human behaviours through the temporal evolution of facial features. Though the RBN have the advantage of being independent of the event temporal scale, the learning is still complex and the difference of representation between the spatial relationships and the temporal ones is not clear.

They are direct acyclic graphs (DAGs) which, based on Bayesian rules, express relations of conditional dependence (through direct arcs) among random variables (nodes). Given two events, A and B, if these are correlated in some way, we can think that knowing one of them which has just happened can improve the knowledge of the other event probability.

Now, we can formalize these networks. Given a probability space (Ω,*A*,P[]), where:

- Ω is the sample space (the collection of the all possible results of the experiment)
- *A* is the Event space (which contains Ω)
- P[] is a probability function with domain in A and codomain in [0,1]

Given two Events, A and B, which belong to *A*, the following expression

$$\mathrm{P[A|B]}$$

represents that A happens knowing that B has just happened; that is the probability of A conditioned by B.

Then, we can define the Bayesian rule: for two events, A and B, belonging to set A, the following relation is valid:

$$P[A|B] = \frac{P[A,B]}{P[B]}, (if P[B] \neq 0) \tag{2.1}$$

where:

$$P[A|B] = P[B|A] * P[A] \tag{2.2}$$

Generalizing to a generic number of elements, we can derive the same conclusions:

Given a probability space (Ω,*A*,P[]), there are:

- $B_1, B_2, ..., B_n$ belonging to *A*
- $\forall i, P[B_i] > 0; i \neq j, B_i B_j = 0; \Omega = \bigcup_i B_i$

For each event A belonging to the set *A*, we have the following relation:

$$P[B_k|A] = \frac{P[A|B_k]P[B_k]}{(\sum_i P[A|B_i]P[B_i])} \tag{2.3}$$

A *Bayesian Network* is a graph for which the following properties are valid:

- A set of random variables represent network nodes;
- A set of edges, having a direction, connect the couples of nodes (the intuitive meaning of an arrow from X node to Y node is that X has a direct influence on Y);
- Each node has a conditional probability table which quantifies the effects that "parents" have on the node, where for parents we want to denote all the nodes whose arrows are going to the considered node;
- The graph has not direct cycles;
- A node that has not direct parents has a table of *marginal probabilities*; if the node is discrete, it contains a probability distribution on the states of the variable that it represents; if the node is continuous, it contains a density gaussian function of the random variable which it represents;
- If a node has no parents, then the node contains a *conditional probability* table.

The graphic structure of a BN allows a non ambiguous representation of inter-dependence among variables, which causes the most important feature of such type of networks: in fact, combined probability distribution X = $\{X_1, X_2, ..., X_n\}$ can be factorized in terms of the product among the network CPTs.

$$P(X = x) = \prod_{i=1}^{n} P(X_i|\pi_i) \tag{2.4}$$

2.1.1.2 HMM (Hidden Markov Models)

A HMM is a probabilistic model in which the system to be modelled is taken as a process of Markov's with unknown parameters that can be obtained from observable parameters, while in a regular Markov model the state can be directly seen by the observer and the only parameters are the probabilities of the transitions among the states, in a *hidden* Markov model the state cannot be directly seen but only the variables affected by the states themselves can be accessed.

Each state has a distribution of probability pertinent to the possible output token. On the other hand the sequence of the tokens generated by a HMM gives some information about the sequence of the states. The principle of this approach is the use of Markovian hypothesis. The probability of being in a given state depends only on the probability of being in the previous one. HMM have been chosen to recognize the American [7] sign language [8] within the limits of oral identification or for identification in writing [9]. Using the HMM, Hongeng [10] [5] has presented an approach to recognize mono-state and multi-state activities in a dynamic scenery. Zhang [11] et al. use them to point out Anomalous Events. The HMM advantage compared with Bayesian nets is the ability to recognize a sequence of events, but are not as convenient for activities that involve more that one person. Coupled Hidden Markov Models have been presented by Brand [12] et al. to reduce this problem, but, once again, they turn out to be inadequate when the activity subjects are more than two, a situation that requires too complex a model and a relevant number of parameters, besides a great learning difficulty during the learning session.

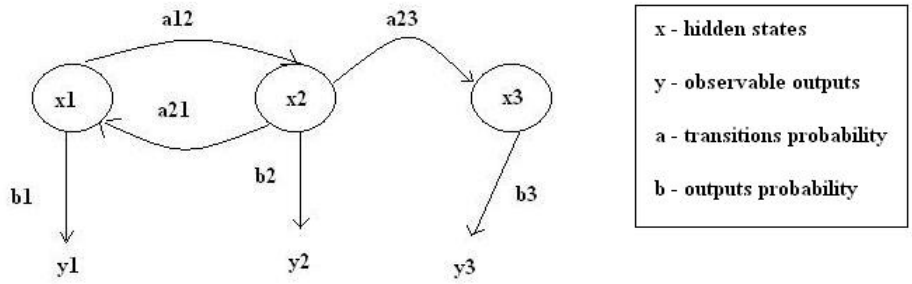


Figure 2.1: Hidden Markov Model

HMM are characterized by the presence of nodes and arcs, as we can see in the 2.2 figure.

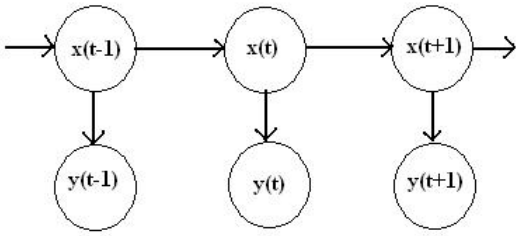


Figure 2.2: Relations among States in Hidden Markov Model

Each node represents a random variable which can accept a certain number of values.

The random variable *x(t)* is the value of the hidden variable at "t" temporal instant. The random variable *y(t)* represents the value of the variable observed at "t" temporal instant. Arrows in the diagram indicate conditional dependencies. It is easy to deduce, observing the diagram, that the value of the hidden variable *x(t)* is only dependent on the value of the hidden variable *x(t-1)* (at "t-1" temporal instant). This theory is called *Markov's theory*. Similarly, the value of the observed variable *y(t)* only depends on the value of the hidden variable *x(t)*.

The probability of an observed sequence Y = y(0),y(1),....,y(L-1) of length L is:

$$P(Y) = \sum_{X} P(Y|X)P(X) \tag{2.5}$$

where the sum involves all the possible hidden node sequences X = x(0),x(1),....,x(L-1)

There are three fundamental problems about HMM:

- Given the model parameters, calculate the probability of a particular output sequence. This problem can be solved through a forward-backward procedure.
- Given the model parameters, find the sequence of hidden nodes which can have probably generated a certain output sequence. This problem can be solved by Viterbi's algorithm.
- Given an output sequence or a collection of sequences, find the set of transitions

among states and output probabilities that have probably caused such outputs. In other words, induce a HMM to establish its parameters for a given sequence of observations. This problem can be resolved by Baum-Welch's algorithm.

2.1.1.3 Relevant Works

There are numerous relevant works in literature following DBN and HMM approaches. For instance, [13] studies how HMMs can be used to recognize complex activites, while [14] and [15] use coupled HMMs. [16] uses Dynamic Bayesian Networks (DBNs) to capture causal relationships between observations and hidden states. [1] developed a stochastic automaton based language to detect activities in video, while [17] presented a HMM-based algorithm. *In contrast, this thesis starts with a set $\mathcal{A}$ of activity models (corresponding to innocuous/dangerous activities) and finds observation sequences that are not sufficiently explained by the models in $\mathcal{A}$.* Such unexplained sequences reflect activity occurrences that differ from the application's expectations.

2.1.2 Learning and then detecting abnormality

2.1.2.1 Pattern Recognition and Machine Learning

The problem of searching for patterns in data is a fundamental one and has a long and successful history. For instance, the extensive astronomical observations of Tycho Brahe in the 16th century allowed Johannes Kepler to discover the empirical laws of planetary motion, which in turn provided a springboard for the development of classical mechanics. Similarly, the discovery of regularities in atomic spectra played a key role in the development and verification of quantum physics in the early twentieth century. The field of pattern recognition is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories. Consider the example of recognizing handwritten digits. Each digit corresponds to a $28 * 28$ pixel image and so can be represented by a vector x comprising 784 real numbers. The goal is to build a machine that will take such a vector x as input and that will produce the identity of the digit $0, \ldots, 9$ as the output. This is a nontrivial problem due to the wide variability of handwriting. It could be tackled using handcrafted rules or heuristics for distinguishing the digits based on the shapes of the strokes, but in practice such an approach leads to a

proliferation of rules and of exceptions to the rules and so on, and invariably gives poor results.

Far better results can be obtained by adopting a machine learning approach in which a large set of *N* digits $x_1, ..., x_N$ called a *training set* is used to tune the parameters of an adaptive model. The categories of the digits in the training set are known in advance, typically by inspecting them individually and hand-labelling them. We can express the category of a digit using *target vector* t, which represents the identity of the corresponding digit.

The result of running the machine learning algorithm can be expressed as a function y(x) which takes a new digit image x as input and that generates an output vector y, encoded in the same way as the target vectors. The precise form of the function y(x) is determined during the *training* phase, also known as the *learning* phase, on the basis of the training data. Once the model is trained it can then determine the identity of new digit images, which are said to comprise a *test set*. The ability to categorize correctly new examples that differ from those used for training is known as *generalization*. In practical applications, the variability of the input vectors will be such that the training data can comprise only a tiny fraction of all possible input vectors, and so generalization is a central goal in pattern recognition. For most practical applications, the original input variables are typically *pre-processed* to transform them into some new space of variables where, it is hoped, the pattern recognition problem will be easier to solve. For instance, in the digit recognition problem, the images of the digits are typically translated and scaled so that each digit is contained within a box of a fixed size. This greatly reduces the variability within each digit class, because the location and scale of all the digits are now the same, which makes it much easier for a subsequent pattern recognition algorithm to distinguish between the different classes. This pre-processing stage is sometimes also called *feature extraction*. Note that new test data must be pre-processed using the same steps as the training data.

Pre-processing might also be performed in order to speed up computation. For example, if the goal is real-time face detection in a high-resolution video stream, the computer must handle huge numbers of pixels per second, and presenting these directly to a complex pattern recognition algorithm may be computationally infeasible. Instead, the aim is to find useful features that are fast to compute, and yet that also preserve useful discriminatory information enabling faces to be distinguished from non-faces. These features

are then used as the inputs to the pattern recognition algorithm. For instance, the average value of the image intensity over a rectangular subregion can be evaluated extremely efficiently, and a set of such features can prove very effective in fast face detection. Because the number of such features is smaller than the number of pixels, this kind of pre-processing represents a form of dimensionality reduction. Care must be taken during pre-processing because often information is discarded, and if this information is important to the solution of the problem then the overall accuracy of the system can suffer.

Applications in which the training data comprises examples of the input vectors along with their corresponding target vectors are known as *supervised learning* problems. Cases such as the digit recognition example, in which the aim is to assign each input vector to one of a finite number of discrete categories, are called *classification* problems. If the desired output consists of one or more continuous variables, then the task is called *regression*. An example of a regression problem would be the prediction of the yield in a chemical manufacturing process in which the inputs consist of the concentrations of reactants, the temperature, and the pressure.

In other pattern recognition problems, the training data consists of a set of input vectors x without any corresponding target values. The goal in such *unsupervised learning* problems may be to discover groups of similar examples within the data, where it is called *clustering*, or to determine the distribution of data within the input space, known as *density estimation*, or to project the data from a high-dimensional space down to two or three dimensions for the purpose of *visualization*.

Finally, the technique of *reinforcement learning* is concerned with the problem of finding suitable actions to take in a given situation in order to maximize a reward. Here the learning algorithm is not given examples of optimal outputs, in contrast to supervised learning, but must instead discover them by a process of trial and error. Typically there is a sequence of states and actions in which the learning algorithm is interacting with its environment. In many cases, the current action not only affects the immediate reward but also has an impact on the reward at all subsequent time steps. For example, by using appropriate reinforcement learning techniques a neural network can learn to play the game of backgammon to a high standard. Here the network must learn to take a board position as input, along with the result of a dice throw, and produce a strong move as the output. This is done by having the network play against a copy of itself for perhaps a million games. A major challenge is that a game of backgammon can involve dozens of

moves, and yet it is only at the end of the game that the reward, in the form of victory, is achieved. The reward must then be attributed appropriately to all of the moves that led to it, even though some moves will have been good ones and others less so. This is an example of a *credit assignment* problem. A general feature of reinforcement learning is the trade-off between *exploration*, in which the system tries out new kinds of actions to see how effective they are, and *exploitation*, in which the system makes use of actions that are known to yield a high reward. Too strong a focus on either exploration or exploitation will yield poor results. Reinforcement learning continues to be an active area of machine learning research. Although each of these tasks needs its own tools and techniques, many of the key ideas that underpin them are common to all such problems.

2.1.2.2 Relevant Works

Several researchers in the literature followed this approach in order to find abnormal activities: so, they usually first learn normal activity models and then detect abnormal/unusual events. [18] suggests a semi-supervised approach to detect abnormal events that are rare, unexpected, and relevant. We do not require "unexplained" events to either be rare or relevant. [19] uses HMMs to detect rare events, while [20] defines an anomaly as an atypical behavior pattern that is not represented by sufficient samples in a training dataset and satisfies an abnormal pattern. [21] defines abnormality as unseen or rarely occurring events—an initial video is used to learn normal behaviors. [22] shows how to detect users with abnormal activities from sensors attached to human bodies. An abnormal activity is defined as "an event that occurs rarely and has not been expected in advance". The same notion of abnormal activity is considered in [23] and [24]. [25] learns patterns of activities over time in an unsupervised way. [26] detects individual anomalies in crowd scenes—an anomaly is defined as a rare or infrequent behavior compared to all other behaviors. Common activities are accepted as normal and infrequent activity patterns are flagged as abnormal. All these approaches first learn normal activity models and then detect abnormal/unusual events. These approaches differ from ours as they consider rare events to be abnormal. In contrast, we consider activities to be unexplained even if they are not rare and the available models are not able to capture them. For example, if a new way to break into cars has occurred many times (and we do not have a model for it), then we want to flag sequences where those activities occur as "unexplained" even if they are not rare. In addition, if a model exists for a rare activity, we

would flag it as “explained”, while many of these frameworks would not.

2.1.3 Similarity-based abnormality.

2.1.3.1 Main features

Another category of approaches able to find unusual activities deals with similarity-based abnormality.

In general, for similarity-based approaches, the main task is to define pairwise distances between all the data points and identify outliers by examining the distance to an examples nearest neighbors. An example is the work by Breunig et al. [27], who applied a density-based clustering algorithm to efficiently detect local outliers. Based on the distance measure and user-defined density thresholds, these algorithms can efficiently detect the occurrence of outliers (or abnormal points) in a high-dimensional space. The basic principle is that if the neighboring points are relatively close, the example is considered normal; otherwise, the example is considered abnormal. The advantage of these approaches is that no explicit distribution needs to be defined to determine outliers and that the methods can be made efficiently for large data sets. However, a difficulty in these approaches lies in the question of how to define effective similarity measures when a large amount of uncertainty exists. For example, in the sensor network area, sensor readings received from sensors vary greatly from time to time, following a stochastic nature. Thus, it would be very difficult to define a distance measure that is sufficiently robust in these settings, making it difficult also to define density measures. Another difficulty lies in the requirement in our problem that the algorithm must be online; that is, efficient models need to be trained ahead of time in order to efficiently detect abnormal events as they occur. Therefore, in summary, when the data do not provide clear-cut shapes, and when the data are stochastic in nature, as in the case of sensor readings, the similarity- based and distance-based approaches cannot work well.

2.1.3.2 Relevant works

[28] proposes an unsupervised technique in which no explicit models of normal activities are built. Each event in the video is compared with all other observed events to determine how many similar events exist. Unusual events are events for which there are no similar events in the video. Hence, this thesis also considers unusual activity as a rare event

and a large number of observations is required to verify if an activity is unusual. [29] uses a similar approach: a scene is considered anomalous when the maximum similarity between the scene and all previously viewed scenes is below a threshold. In [30], frequently occurring patterns are normal and patterns that are dissimilar from most patterns are anomalous. [31] learns trajectory prototypes and detects anomalous behaviors when visual trajectories deviate from the learned representations of typical behaviors. An unsupervised approach, where an abnormal trajectory refers to something that has never (or rarely) seen, has been proposed in [32]. A normal trajectory is intended to be one similar enough to one or more trajectories that the system already knows. In [13], activities performed by a group of moving and interacting objects are modeled as shapes and abnormal activities are defined as a change in the shape activity model. In the context of elder care, [33] proposes an approach that first analyzes and designs features, and then detects abnormal activities using a method based on the designed features and Support Vector Data Description. [34] proposes a methodology to characterize novel scenes over long time periods without a priori knowledge. A hierarchical modeling process, characterizing an activity at multiple levels of resolution, is developed to classify and predict future activities and detect abnormal behavior.

2.1.4 Other relevant work

In [35], unusual events are detected by monitoring the scene with *monitors* which extract local low-level observations from the video stream. The monitor computes the likelihood of a new observation with respect to the probability distribution of prior observations. If the likelihood falls below a threshold, then the monitor outputs an alert. The local alerts issued by the monitors are then combined. [36] automatically learns high frequency events (taking spatio-temporal aspects into account) and declares them normal—events deviating from these rules are anomalies. [37] learns storylines from weakly labeled videos. A storyline includes the actions that occur in a video and their causal relationships. AND-OR graphs are used to represent storyline models.

The first preliminary notion of unexplained activities used in this thesis has been introduced in [38].

2.2 Cyber Security

Intrusion detection and alert correlation techniques provide valuable and complementary tools for identifying and monitoring security threats in complex network infrastructures. Intrusion detection systems (IDS) can monitor network traffic for suspicious behavior and trigger security alerts accordingly [39], [40], [41]. Alert correlation methods can aggregate such alerts into multi-step attack scenarios [42], [43], [44], [39], [45]. Intrusion detection has been studied for about thirty years, since it was first identified in the Anderson report [46], and it is based on the assumption that an intruder's behavior will be noticeably different from that of a legitimate user and that many unauthorized actions are detectable [39].

2.2.1 Intrusion detection

The historical evolution of computer systems along with the Internet has drastically reduced the cost of transporting information over the world thanks to more work being done online. With the apparent effectiveness of internet working, today's networked information systems are expanding ubiquitously. Indeed, recent years have seen the explosive growth in the number of devices interconnected to computer and telecommunication networks. As a result, tremendous amounts of data are transmitted from and to, processed, and stored at central networked information systems. Without doubts, the networked information systems play crucial roles for most governments, enterprises, and even individuals. Therefore, they must remain not only up-and running but also secure against unwanted harmful actions such as attack, misuse, and abuse. However, the information systems have witnessed ever-increasing instances of malicious activities despite the existence of a variety of security technologies. Attackers attempt to steal, modify, and destroy valuable information and at worst damage the victim systems. And also, attackers often attempt to make services merely unavailable to intended legitimate users by exhausting the system resources DoS/DDoS. In many systems, security cannot allow these threats because the impact of such attacks is immeasurable and irrevocable. Despite of the existence of a variety of security measures, attackers eventually manage to get through them and this helps attacking techniques to speedily evolve. Thus, even if a security system fails to defend against an attack, it should be well aware of being attacked and have a mechanism to perform countermeasures in order to prevent further

attacks and reduce the damage and loss resulting from the attack. That is the main aim of Intrusion Detection Systems (IDS).

Thus, intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices. Intrusion prevention is the process of performing intrusion detection and attempting to stop detected incidents happening again. Intrusion detection and prevention systems (IDPS) are primarily focused on identifying possible incidents, logging information about them, attempting to stop them, and reporting them to security administrators. In addition, organizations use IDPSs for other purposes, such as documenting existing threats, identifying problems with security policies, and deterring individuals from violating them. IDPSs have become a necessary addition to the security infrastructure of nearly every organization that relies on information technology.

Intrusion detection has been an active field of research for about three decades, starting in 1980 with the publication of John Andersons report, *Computer Security Threat Monitoring and Surveillance*. In an informatic system, says Anderson, an intrusion is any attempt to breach security, dependability and availability of this system.

Intrusion detection is based on the assumption that an intruder's behavior will be noticeably different from that of a legitimate user and that many unauthorized actions are detectable. Infact, Anderson states [46]: "Intrusion detection systems analize information about the activity performed in a computer system or network, looking for evidence of malicous behavior", that is to say an IDS is a system that detects unauthorized access or potential attacks on informatic systems through informations source available on the system (log) or on the network (network traffic). In addition, IDSs are systems used mainly for monitoring network traffic through a set of rules and flexible algorithms in order to detect attacks on the autonomous system (AS) involved. In figure 2.3 the architecture of a generic IDS is reported , and its principal component are shown:

- Data collection: in this phase the traffic envolved on interested system is collected and stored.
- Detection: in this step the data are analyzed and a subset of them are detected according to state information and detection policy.
- Response: in this step the output of IDS is built according to response policy.

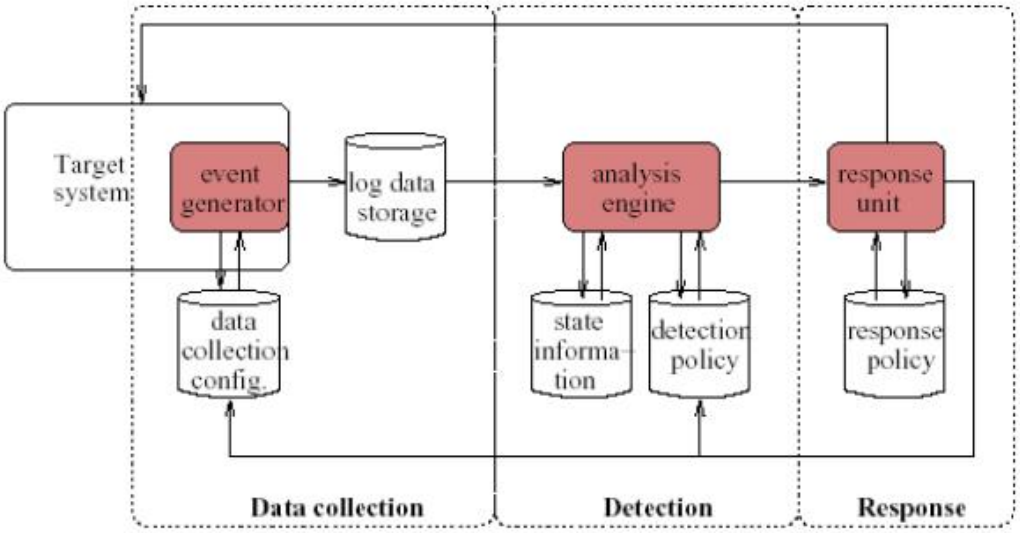


Figure 2.3: IDS architecture

Compared with the modality of analysis selected, it is possible to distinguish two types of intrusion detection systems [47]:

Signature based: An IDS is called *signature based* [40] if it uses a knowledge base in order to detect an attack. An IDS analyzes the collected data and compares them to a set of attack signatures to discover any anomaly actions. So, an anomaly action is discovered if it exists by a correspondence between its signature and that of a known attack, stored in a knowledge base. A *signature* is a pattern that corresponds to a known threat while *signature-based detection* is the process of comparing signatures against observed events to identify possible incidents. Examples of signatures are the following:

- A telnet attempt with a username of root is a violation of an organization's security policy.
- An e-mail with a subject of Free pictures and an attachment filename of freepics.exe, which are characteristics of a known form of malware.
- An operating system log entry with a status code value of 645, which indicates that the hosts auditing has been disabled.

Usually, an attack signature belongs to only one attack and it is possible that an attack

signature can be associated with more than one attack. On the one hand the simplicity of the approach and the possibility to use established techniques and optimized pattern matching make the use of IDS (misuses based) preferable; on the other hand, its quality depends only on set of the signature compared.

Anomaly based: An anomaly based IDS [40] tries to build correct models of resources' normal activity and stores them in a knowledge base. This IDS detects anomaly activities through a comparison of the stored actions with another knowledge base of known threats; quantitatively, an action is considered important if some features of actions exceed the appropriate thresholds. Thus, *Anomaly-based detection* is the process of comparing definitions of what activity is considered normal against observed events to identify significant deviations. This IDS uses profiles that represent the normal behavior of such things as users, hosts, network connections, or applications. The profiles are developed by monitoring the characteristics of typical activity over a period of time. An initial profile is generated over a period of time (typically days or weeks) sometimes called a training period. Profiles for anomaly-based detection can either be static or dynamic. Once generated, a static profile is unchanged unless the IDS is specifically directed to generate a new profile. A dynamic profile is adjusted constantly as additional events are observed. Because systems and networks change over time, the corresponding measures of normal behavior also change; a static profile will eventually become inaccurate, so it needs to be regenerated periodically. Dynamic profiles do not have this problem, but they are susceptible to evasion attempts from attackers. For example, an attacker can perform small amounts of malicious activity occasionally, then slowly increase the frequency and quantity of activity. If the rate of change is sufficiently slow, the IDS might think the malicious activity is normal behavior and include it in its profile. So, the anomaly based IDS depends on the knowledge of the attacks. This is a robust approach and very effective at detecting previously unknown threats, but it is more difficult to research good model in order to describe normal actions in heterogeneous contexts.

An other classification of IDS based on the type of sensor is the following:

Network-based: The sensors are placed at strategic points on the internal network and it is very important that the sniffer can be as invisible and transparent to the attacker which, otherwise, could take countermeasures, which is why the network interfaces of the sensors are configured in stealth mode that operate without IP address, and without sending any traffic. These IDSs have the inherent advantage of being independent from

the operating system because they need packages coming from the network and not a resource or as a specific machine. Moreover, their development will not impact the structure of the existing network as it is to place the sensor (usually a purely passive) in strategic nodes. This causes the operating costs are relatively low, as there are few modules to maintain. However, this apparent advantage has also its drawbacks because these IDSs can only detect attacks visible on the network segment. If for example, the signature of a certain attack was encrypted, then the IDS would not be able to find it. From the performance point of view, if these systems need to try to detect fragmented attacks, the complexity required by the reconstruction is a theoretical problem rather than a real one (this operation being feasible in theory only, with a processor to arbitrary power).

Host-based: These systems analyze activities with great level of detail and accuracy to determine the exact processes involved in a particular attack. Unlike network-based IDS, host-based ones can directly observe the effects of an intrusion since they have access and monitor the files and processes that are normally the intended target of such attacks. When implementing a host-based IDS, being strongly dependent on the operating system, development and deployment costs are higher than a network-based IDS ones, as it requires the installation and configuration on multiple machines, often heterogeneous. This is ill-suited to today's network based computer systems, deep and highly distributed with equal information sources.

Finally, we can classify the IDS in:

On-line: systems that analyze the data while actions occur. They are called real-time to indicate that the IDS is designed to handle a certain stream of data without losing packets, and without getting a buffer waiting to be verified.

Off-line: systems that analyze the data retrospectively. The general tendency is to consider the latter more as a tool of computer forensics, that is the verification of intrusion with general purposes of the judicial police, then intrusion detection.

A relatively innovation in this area are in-line systems that are often called Intrusion Prevention System (IPS). In essence, if the IDS is in-line, it is positioned as a firewall or a switch on the path of the packages and its main function is to be able to act as a filter as well as an analyzer, while the traditional IDS are usually placed on a network port that has a copy of all traffic.

In summary, signature-based methods have been used extensively to detect malicious

activities. On the other hand, in profile-based methods, a known deviation from the norm is considered anomalous (e.g. HTTP traffic on a non-standard port).

In contrast, in this thesis, we consider the case where we have a set $\mathcal{A}$ of known activities (both innocuous and dangerous)—and we are looking for observation sequences that cannot be explained by either (if they were, they would constitute patterns that were known a priori). These need to be flagged as they might represent "zero day" attacks—attacks that were never seen before and vary significantly from past known access patterns.

2.2.2 Correlation techniques

Since Intrusion Detection Systems (IDS) are increasingly deployed in the network, they could generate an overwhelming number of alerts with true alerts mixed with false ones. Most of these alerts are raised independently, making it very hard for the network administrator or intrusion response system to understand the real security situation of the network and provide response to the intrusions. Consequently, alert correlation has become a critical process in intrusion detection and response.

Alert correlation [43] can be very beneficial especially for intrusion response. Firstly, it reduces the volume of alerts that needs to be handled. IDSs may generate thousands of alerts per day. One of the main tasks of alert correlation is to aggregate duplicate alerts and filter low-interest alerts. After these steps, the number of alerts that are presented to network administrators will be greatly reduced.

Secondly, due to the problem of false positives, it is impossible to respond to every alert that is reported by IDS. Only those which are detected with high confidence will be considered for response action. Alert correlation provides a way to increase the detection confidence. Generally speaking, correlated alerts are less likely to be false alerts, because they suggest the preparation or continuation of attacks. It is rare for a legitimate user to trigger multiple alerts, and at the same time, these alerts are correlated as different stages of an attack. On the other hand, if we get multiple correlated alerts, and they fall into different stages of an attack, such as scanning, remote to local, denial of service (DoS), then the possibility of these alerts being true alerts should be higher than the case that these alerts are independent. Normally, for attacks that have great security impact on the protected network such as DoS and worms, response actions will be taken without observing any correlated alerts, because these kinds of attack themselves are not likely

to be false alerts due to their complexity. Moreover, their severity is high and therefore should be handled with high priority. However, there are cases where certain types of alerts can have a high false positive rate, such as password-guessing. In that case, if a response unit receives such an uncorrelated alert, it knows that it could be a false alert, and considering its severity, it might choose not to respond at this time. But, if it receives this alert right after it receives a port scan alert from the same source targeting the same host, it will have higher confidence that this is a true alert and therefore take appropriate response actions.

Thirdly, correlated alerts provide a succinct, high-level view of the security state of the network under surveillance. By knowing the security state, the network administrator can make an appropriate plan to respond to intrusions. Consider a situation in which one single host in the network is infected by a certain type of worm, and the situation that multiple hosts in different subnets are infected in a short period of time indicating the spreading of the worm. The response action with respect to these two different situations can be totally different: In the former case, he might just need to disconnect the host from the network, and remove the malicious program. In the latter case, he might also need to modify the firewall rules to block the corresponding type of traffic in order to prevent the worm from spreading outside the network.

Another important use of alert correlation is to recognize the strategies or plans of different intrusions and infer the goal of attacks. Suppose that the next step or the final goal of an attacker can be identified by looking at the pattern of the intrusive behavior, we can take actions to prevent the attack from escalating and therefore minimize the damage to the asset. Alert correlation provides some means to group different logically-connected alerts into attack scenarios.

Alert correlation is defined as a process that contains multiple components with the purpose of analyzing alerts and providing high-level insight on the security state of the network under surveillance.

In summary, the goal of correlation is to find causal relationships between alerts in order to reconstruct attacks from isolated alerts. This goal is achieved by providing a higher level view of the actual attacks [44, 45, 39, 48, 49, 50].

From a conceptual point of view, both intrusion detection systems and alert correlation methods aggregate fine grained information into higher level views of the attack, although they operate at different levels of abstraction, as shown in Figure 2.4. Moreover,

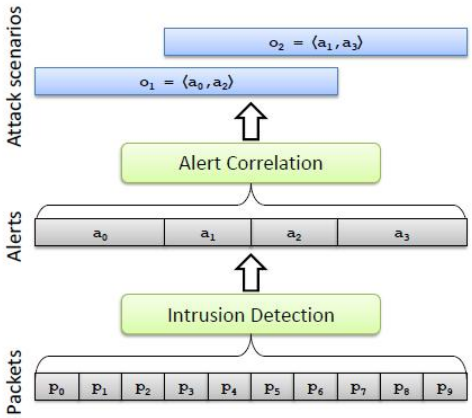


Figure 2.4: Conceptual diagram in cyber security domain

both rely on models encoding a priori knowledge of either normal or malicious behavior, and cannot appropriately deal with events that are not explained by the underlying models. In practice, all these methods are incapable of quantifying how well available models explain a sequence of events observed in data streams (data packets and alerts respectively) feeding the two classes of tools.

However, the framework and algorithms proposed for identifying unexplained activities are domain independent and may be applied to any domain including both activity detection in video and in cyber-security.

Chapter 3

Activity Detection Models

3.1 Basic Activity Model

The aim of this section is to define a new method to model well-known activities. Such a goal has been achieved by extending the stochastic activity model of [1] adding a function δ which expresses a constraint on the maximum "temporal distance" between two actions in an activity (though we make no claims of novelty for this).

We assume the existence of a finite set $\mathcal{S}$ of *action symbols*, corresponding to observable atomic actions. For instance, in the video domain, action symbols might be recognized by sophisticated image processing algorithms, while in the cyber-security domain, they may simply be read from a log file. Though our unexplained sequence detection framework is domain-independent, in some domains such as video surveillance, the problem of recognizing low-level actions in video can be a big challenge: for this reason, we embedded the *Unexplained Activity Detector* into complete prototype implementations we designed and implemented (starting from the output of cameras/sensors for the video surveillance context and from the packets for the cyber security one) in order to make a complete experimentation on real-world datasets on both the considered contexts. We will describe in details the whole followed process in Chapter 5.

Definition 3.1.1 (Stochastic activity) A stochastic activity *is a labeled directed graph* $A = (V, E, \delta, \rho)$ *where*

- V *is a finite set of nodes labeled with action symbols from* $\mathcal{S}$*;*

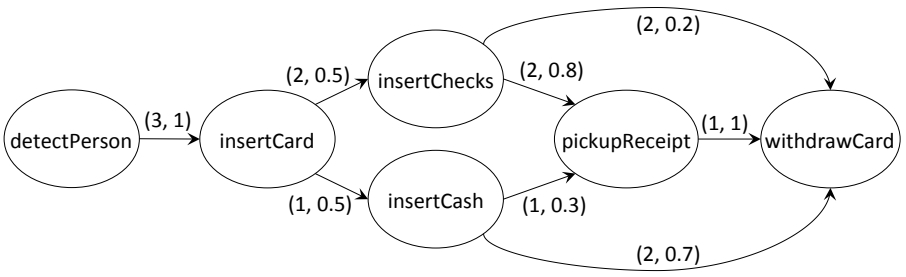


Figure 3.1: Example of stochastic activity: ATM deposit

- $E \subseteq V \times V$ *is a set of edges;*
- $\delta : E \to \mathbb{N}^+$ *associates, with each edge* $\langle v_i, v_j \rangle$*, an upper bound on the time that can elapse between* v_i *and* v_j*;*
- ρ *is a function that associates, with each node* $v \in V$ *having out-degree 1 or more, a probability distribution on* $\{\langle v, v' \rangle \mid \langle v, v' \rangle \in E\}$*, i.e.,* $\sum_{\langle v,v' \rangle \in E} \rho(\langle v, v' \rangle) = 1$*;*
- *there exists at least one* start node *in the activity definition, i.e.* $\{v \in V \mid \nexists\, v' \in V \; s.t. \; \langle v', v \rangle \in E\} \neq \emptyset$*;*
- *there exists at least one* end node *in the activity definition, i.e.* $\{v \in V \mid \nexists\, v' \in V \; s.t. \; \langle v, v' \rangle \in E\} \neq \emptyset$.

Figure 3.1 shows a stochastic activity of deposits at an Automatic Teller Machine (ATM). Each edge e is labeled with $(\delta(e), \rho(e))$. For an edge $e = \langle s_1, s_2 \rangle$, $\delta(e)$ specifies the maximum time between when s_1 is observed and when s_2 is observed. ρ specifies the probability of going from one node to another (the probability distribution associated with a node gives the transition probability from that node). For instance, the two edges starting at node insertCard mean that there is a 50% probability of going to node insertChecks and a 50% probability of going to node insertCash from node insertCard. In addition, insertChecks and insertCash must follow insertCard within 2 and 1 time units, respectively.

In general, each node of a stochastic activity definition is something that can be detected by application code. For instance, if we are tracking activities in video, each node in a stochastic activity would be something that can be detected by an image processing

program, e.g. "Detect Person" in Figure 3.1 may be identified as holding only if a probabilistic face recognition program returns a probability over some threshold that a given frame (or block of frames) contains a face in it. Likewise, in a cybersecurity application, a node in an activity such as "Attempted login" may only be identified as occurring if a log file archives a login attempt. For the sake of simplicity, we use "high level" descriptions of nodes in our examples, as opposed to low level descriptions (e.g., the color histogram of a given image shows over 70% of the colors are a certain shade).

An instance of a stochastic activity A is a path in A from a start node to an end node.

Definition 3.1.2 (Stochastic activity instance) *An* instance *of a stochastic activity* (V, E, δ, ρ) *is a sequence* $\langle s_1, \ldots, s_m \rangle$ *of nodes in* V *such that*

- $\langle s_i, s_{i+1} \rangle \in E$ *for* $1 \leq i < m$*;*
- $\{s \mid \langle s, s_1 \rangle \in E\} = \emptyset$*, i.e.,* s_1 *is a start node; and*
- $\{s \mid \langle s_m, s \rangle \in E\} = \emptyset$*, i.e.,* s_m *is an end node.*

The probability of the instance is $\prod_{i=1}^{m-1} \rho(\langle s_i, s_{i+1} \rangle)$.

In Figure 3.1, $\langle$detectPerson, insertCard, insertCash, withdrawCard$\rangle$ is an instance with probability 0.35. Throughout the thesis, we assume an arbitrary but fixed set $\mathcal{A}$ of stochastic activities.

The preceding definitions do not take observation sequences into account. In order to define when activity occurrences are detected in a sequence of time-stamped observation data, we first need to formally define an observation sequence. An *observation sequence* is a finite sequence of *observation IDs*. An observation ID (OID) f has an associated timestamp, denoted $f.ts$, and an associated set of action symbols, denoted $f.obs$. Without loss of generality, we assume timestamps to be positive integers. For instance, if our observation sequence is a video, then the OIDs may be frame IDs with $f.ts$ being the timestamp associated with frame f and $f.obs$ being the actions detected in frame f. On the other hand, if our observation sequence is a sequence of transactions at a website, the OIDs are transaction IDs, $f.ts$ is the timestamp associated with transaction f, and $f.obs$ are the actions associated with transaction f.

Example 3.1.1 (Video example) *An observation sequence might be a video* $v = \langle f_1, f_2, f_3, f_4, f_5 \rangle$*, where the* f_i*'s are frame IDs,* $f_i.ts = i$ *for* $1 \leq i \leq 5$*,* $f_1.obs = \{\textsf{detectPerson}\}$*,* $f_2.obs = \{\textsf{insertCard}\}$*,* $f_3.obs = \{\textsf{insertCash}\}$*,* $f_4.obs = \{\textsf{withdrawCash}\}$*,* $f_5.obs = \{\textsf{withdrawCard}\}$*. Notice that* withdrawCash *in frame* f_4 *does not appear in the stochastic activity of Figure 3.1. In general, action symbols may be detected in a frame even if they do not appear in the definition of a stochastic activity because it is irrelevant for that activity.*

Throughout the thesis, we use the following terminology and notation for (general) sequences. Suppose $S_1 = \langle a_1, \ldots, a_n \rangle$ and $S_2 = \langle b_1, \ldots, b_m \rangle$ are two sequences. S_2 is a *subsequence* of S_1 iff there exist $1 \leq j_1 < j_2 < \cdots < j_m \leq n$ s.t. $b_i = a_{j_i}$ for $1 \leq i \leq m$. If $j_i = j_{i+1} - 1$ for $1 \leq i < m$, then S_2 is a *contiguous* subsequence of S_1. We write $S_1 \cap S_2 \neq \emptyset$ iff S_1 and S_2 have a common element and write $e \in S_1$ iff e is an element appearing in S_1. The *concatenation* of S_1 and S_2, i.e., the sequence $\langle a_1, \ldots, a_n, b_1, \ldots, b_m \rangle$, is denoted by $S_1 \cdot S_2$. Finally, $|S_1|$ denotes the number of elements in S_1.

We now define an occurrence of a stochastic activity in an observation sequence.

Definition 3.1.3 (Activity occurrence) *Let* v *be an observation sequence and* $A = (V, E, \delta, \rho)$ *a stochastic activity. An* occurrence *o of A in v is a sequence* $\langle (f_1, s_1), \ldots, (f_m, s_m) \rangle$ *such that*

- $\langle f_1, \ldots, f_m \rangle$ *is a subsequence of* v*,*
- $\langle s_1, \ldots, s_m \rangle$ *is an instance of* A*,*
- $s_i \in f_i.obs$*, for* $1 \leq i \leq m$*, and* [1]
- $f_{i+1}.ts - f_i.ts \leq \delta(\langle s_i, s_{i+1} \rangle)$*, for* $1 \leq i < m$*.*

The probability of o, denoted $p(o)$*, is the probability of the instance* $\langle s_1, \ldots, s_m \rangle$*.*

When concurrently monitoring multiple activities, shorter activity instances generally tend to have higher probability. To remedy this, we normalize occurrence probabilities by introducing the relative probability $p^*(o)$ of an occurrence o of activity A as $p^*(o) = \frac{p(o)}{p_{max}}$, where p_{max} is the highest probability of any instance of A.

[1] With a slight abuse of notation, we use s_i to refer to both node s_i and the action symbol labeling it.

Example 3.1.2 (Video example) *Consider the video of Example 3.1.1. An occurrence of the activity of Figure 3.1 is* $o = \langle (f_1, \textsf{detectPerson}), (f_2, \textsf{insertCard}), (f_3, \textsf{insertCash}), (f_5, \textsf{withdrawCard}) \rangle$*, and* $p^*(o) = 0.875$*. Notice that if the edge going from* insertCash *to* withdrawCard *was labeled with 1 by* δ*, then* o *would not have been an activity occurrence because* withdrawCard *was required to follow* insertCash *within at most 1 time unit, whereas it occurs after 2 time units in the video.*

We use $\mathcal{O}(v)$ to denote the set of all activity occurrences in v. Whenever v is clear from the context, we write $\mathcal{O}$ instead of $\mathcal{O}(v)$.

Next section (3.2) describes our framework for discovering unexplained activities in an application-independent manner. It is worth noting that the actual input of the framework consists of an observation sequence and a set of activity occurrences (each with a probability). Though our framework is domain-independent, there can be challenges in providing the observations associated with OIDs in some domains (e.g. video surveillance, where identifying the low level actions in a video frame can be highly non-trivial): we have built a specific and complete prototype implementation to address this problem in video surveillance context, as described in Chapter 5.

3.2 UAP Model

This section defines the probability that an observation sequence is unexplained by $\mathcal{A}$. We note that the occurrence of an activity in an observation sequence can involve conflicts. For instance, consider the activity occurrence o in Example 3.1.2 and suppose there is a second activity occurrence o' such that $(f_1, \textsf{detectPerson}) \in o'$. In this case, there is an implicit conflict because $(f_1, \textsf{detectPerson})$ belongs to both occurrences, but in fact, detectPerson can only belong to one activity occurrence, i.e. though o and o' may both have a non-zero probability, the probability that these two activity occurrences coexist is 0. Formally, we say two activity occurrences o, o' *conflict*, denoted $o \nsim o'$, iff $o \cap o' \neq \emptyset$. We now use this to define possible worlds.

Definition 3.2.1 (Possible world) *Let* $\mathcal{O}$ *be the set of all activity occurrences in an observation sequence* v*. A* possible world *for* v *is a subset* w *of* $\mathcal{O}$ *s.t.* $\nexists o_i, o_j \in w, o_i \nsim o_j$.

Thus, a possible world is a set of activity occurrences which do not conflict with one another, i.e., an action symbol in an OID cannot belong to two distinct activity occurrences in the same world. We use $\mathcal{W}(v)$ to denote the set of all possible worlds for an observation sequence v; whenever v is clear from the context, we simply write $\mathcal{W}$.

Example 3.2.1 (Video example) *Consider a video with two conflicting occurrences o_1, o_2. There are 3 possible worlds: $w_0 = \emptyset$, $w_1 = \{o_1\}$, and $w_2 = \{o_2\}$. Note that $\{o_1, o_2\}$ is not a world as $o_1 \nsim o_2$. Each world represents a way of explaining what is observed. The first world corresponds to the case where nothing is explained, the second and third worlds correspond to the scenarios where we use one of the two possible occurrences to explain the observed action symbols.*

Note that any subset of $\mathcal{O}$ not containing conflicting occurrences is a legitimate possible world—possible worlds are not required to be maximal w.r.t. $\subseteq$. In the above example, the empty set is a possible world even though there are two other possible worlds $w_1 = \{o_1\}$ and $w_2 = \{o_2\}$ which are supersets of it. The reason is that o_1 and o_2 are uncertain, so the scenario where neither o_1 nor o_2 occurs is a legitimate one. We illustrate this below.

Example 3.2.2 (Video example) *Suppose we have a video where a single occurrence o has $p^*(o) = 0.6$. In this case, it is natural to say that there are two possible worlds $w_0 = \emptyset$ and $w_1 = \{o\}$ and expect the probabilities of w_0 and w_1 to be 0.4 and 0.6, respectively. By restricting ourselves to maximal possible worlds only, we would have only one possible world, w_1, whose probability is 1, which is wrong.*

It is worth noting that the problem of finding possible worlds corresponds to the problem of finding the independent sets of a graph: occurrences are vertices, conflicts are edges, possible worlds are independent sets. Thus, algorithms to find maximal independent sets can be directly applied to compute possible worlds—all possible worlds can be simply obtained by taking all subsets of the maximal independent sets. An efficient algorithm for generating all the maximal independent sets has been proposed in [51], processing time and memory space are bounded by $O(nm\mu)$ and $O(n + m)$, respectively, where n, m, and μ are the numbers of vertices (occurrences in our case), edges

(conflicts in our case), and maximal independent sets (possible worlds in our case) of a graph.

We use $\overset{*}{\nsim}$ to denote the transitive closure of $\nsim$. Clearly, $\overset{*}{\nsim}$ is an equivalence relation and determines a partition of $\mathcal{O}$ into equivalence classes $\mathcal{O}_1, \ldots, \mathcal{O}_m$. Here the basic idea is to partition the observation sequence into subsequences containing occurrences that conflict directly or in a "transitive" way (we will formally define this with the notion of a *Conflict-Based Partitioning* in Definition 3.2.2 and illustrate it in Example 3.2.4). Equivalence classes that temporally overlap are collapsed into a single one. Two equivalence classes $\mathcal{O}_i$ and $\mathcal{O}_j$ temporally overlap iff $[min(\mathcal{O}_i), max(\mathcal{O}_i)] \cap [min(\mathcal{O}_j), max(\mathcal{O}_j)] \neq 0$, where $min(\mathcal{O}_i) = \min\{f.ts \mid \exists o \in \mathcal{O}_i, (f, s) \in o\}$, $max(\mathcal{O}_i) = \max\{f.ts \mid \exists o \in \mathcal{O}_i, (f, s) \in o\}$, and $min(\mathcal{O}_j)$, $max(\mathcal{O}_j)$ are analogously defined.

Example 3.2.3 (Video example) *Suppose we have a video $v = \langle f_1, \ldots, f_{16} \rangle$ s.t. five occurrences o_1, o_2, o_3, o_4, o_5 are detected as depicted in Figure 3.2, that is, $o_1 \nsim o_2$, $o_2 \nsim o_3$, and $o_4 \nsim o_5$. There are two equivalence classes determined by $\overset{*}{\nsim}$, namely $\mathcal{O}_1 = \{o_1, o_2, o_3\}$ and $\mathcal{O}_2 = \{o_4, o_5\}$.*

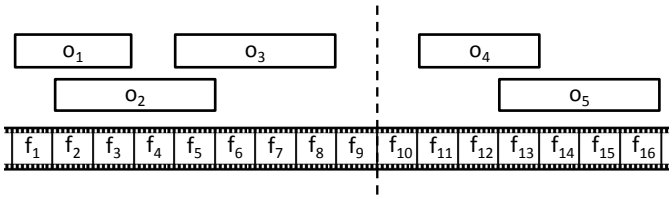


Figure 3.2: Conflict-Based Partitioning of a video

The equivalence classes determined by $\overset{*}{\nsim}$ lead to a conflict-based partitioning of an observation sequence.

Definition 3.2.2 (Conflict-Based Partitioning) *Let v be an observation sequence and $\mathcal{O}_1, \ldots, \mathcal{O}_m$ the equivalence classes determined by $\overset{*}{\nsim}$. A* Conflict-Based Partitioning *(CBP) of v is a sequence $\langle v_1, \ldots, v_m \rangle$ such that:*

- $v_1 \cdot \cdots \cdot v_m = v$, *and*
- $\mathcal{O}(v_i) = \mathcal{O}_i$, *for* $1 \leq i \leq m$.

The v_i's are called segments.

Example 3.2.4 (Video example) *A CBP of the video in Example 3.2.3 is $\langle v_1, v_2\rangle$, where $v_1 = \langle f_1, \ldots, f_9\rangle$ and $v_2 = \langle f_{10}, \ldots, f_{16}\rangle$. Another partitioning of the same video is the one where $v_1 = \langle f_1, \ldots, f_{10}\rangle$ and $v_2 = \langle f_{11}, \ldots, f_{16}\rangle$.*

Thus, activity occurrences determine a set of possible worlds (different ways of explaining an observation sequence). We wish to find a probability distribution over all possible worlds that (i) is consistent with the relative probabilities of the occurrences, and (ii) takes conflicts into account. We assume the user specifies a function $Weight : \mathcal{A} \rightarrow \mathbb{R}^+$ which assigns a weight to each activity and prioritizes the importance of the activity. For instance, highly threatening activities may be assigned a high weight. The weight of an occurrence o of activity A is the weight of A. We use $C(o)$ to denote the set of occurrences conflicting with o, i.e., $C(o) = \{o' \mid o' \in \mathcal{O} \wedge o' \nsim o\}$. Note that $o \in C(o)$; and $C(o) = \{o\}$ when o does not conflict with any other occurrence. Finally, we assume that activity occurrences belonging to different segments are independent events. Suppose p_w denotes the (unknown) probability of world w. As we know the probability of occurrences, and as each occurrence occurs in certain worlds, we can induce a set of nonlinear constraints that will subsequently be used to learn the values of the p_w's.

Definition 3.2.3 *Let v be an observation sequence and $\mathcal{O}_1, \ldots, \mathcal{O}_m$ the equivalence classes determined by $\overset{*}{\nsim}$. We define the non-linear constraints $NLC(v)$ as follows:*

$$\begin{cases} p_w \geq 0, \quad \forall w \in \mathcal{W} \\ \sum\limits_{w \in \mathcal{W}} p_w = 1 \\ \sum\limits_{w \in \mathcal{W}\ s.t.\ o \in w} p_w = p^*(o) \cdot \dfrac{Weight(o)}{\sum_{o_j \in C(o)} Weight(o_j)}, \forall o \in \mathcal{O} \\ p_w = \prod\limits_{k=1}^{m} \sum\limits_{w' \in \mathcal{W}\ s.t.\ w' \cap \mathcal{O}_k = w \cap \mathcal{O}_k} p_{w'} \qquad \forall w \in \mathcal{W} \end{cases}$$

The first two types of constraints enforce a probability distribution over the set of possible worlds. The third type of constraint ensures that the probability of occurrence o—which is the sum of the probabilities of the worlds containing o—is equal to its relative

probability $p^*(o)$ weighted by $\frac{Weight(o)}{\sum_{o_j \in C(o)} Weight(o_j)}$. Note that: (i) the value on the right-hand side of the third type of constraint decreases as the amount of conflict increases, (ii) if an occurrence o is not conflicting with any other occurrence, then its probability $\sum_{w \in \mathcal{W}\ s.t.\ o \in w} p_w$ is equal to $p^*(o)$, i.e., the probability returned by the stochastic automaton. The last kind of constraint reflects independence between segments. In general $NLC(v)$ might admit multiple solutions.

Example 3.2.5 (Video example) *Consider a single-segment video consisting of frames $f_1, \ldots, f_9$ (cf. Figure 3.2). Suppose o_1, o_2, o_3 have been detected with relative probabilities 0.3, 0.6, and 0.5, respectively. Suppose the weights of o_1, o_2, o_3 are 1, 2, 3, respectively. Five worlds are possible: $w_0 = \emptyset$, $w_1 = \{o_1\}$, $w_2 = \{o_2\}$, $w_3 = \{o_3\}$, and $w_4 = \{o_1, o_3\}$. Then, $NLC(v)$ is:*

$$
\begin{aligned}
& p_i \geq 0 \qquad 0 \leq i \leq 4 \\
& p_0 + p_1 + p_2 + p_3 + p_4 = 1 \\
& p_1 + p_4 = 0.3 \cdot \tfrac{1}{3} \\
& p_2 = 0.6 \cdot \tfrac{1}{3} \\
& p_3 + p_4 = 0.5 \cdot \tfrac{3}{5}
\end{aligned}
$$

which has multiple solutions.

For brevity, we do not explicitly list the independence constraints.

One solution is $p_0 = 0.4$, $p_1 = 0.1$, $p_2 = 0.2$, $p_3 = 0.3$, $p_4 = 0$. Another solution is $p_0 = 0.5$, $p_1 = 0$, $p_2 = 0.2$, $p_3 = 0.2$, $p_4 = 0.1$.

In the rest of the thesis, we assume that $NLC(v)$ is solvable. This can be easily checked via both a non-linear constraint solver, as well as methods developed in next subsection (3.2.1). We say that a sequence $S = \langle (f_1, s_1), \ldots, (f_n, s_n) \rangle$ *occurs in* an observation sequence v iff $\langle f_1, \ldots, f_n \rangle$ is a contiguous subsequence of v and $s_i \in f_i.obs$ for $1 \leq i \leq n$. We give two semantics for S to be unexplained in a world $w \in \mathcal{W}$. Intuitively, S is *totally* (resp. *partially*) unexplained in w iff w does not explain every (resp. at least one) symbol of S. More formally:

1. S is *totally unexplained* in w, denoted $w \nvDash_T S$, iff $\forall (f_i, s_i) \in S, \nexists o \in w, (f_i, s_i) \in o$;
2. S is *partially unexplained* in w, denoted $w \nvDash_P S$, iff $\exists (f_i, s_i) \in S, \nexists o \in w, (f_i, s_i) \in o$.

Example 3.2.6 (Video example) *Suppose we have a video $v = \langle f_1, \ldots, f_9 \rangle$ such that $f_i.obs = \{s_i\}$, $1 \leq i \leq 9$, and two occurrences o_1 and o_2 are detected (cf. Figure 3.3).*

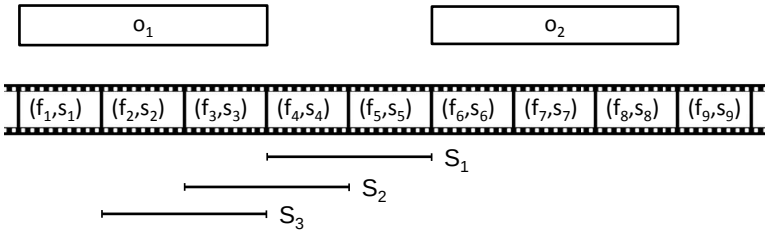


Figure 3.3: Totally and partially unexplained sequences

The four possible worlds are: $w_0 = \emptyset$, $w_1 = \{o_1\}$, $w_2 = \{o_2\}$, $w_3 = \{o_1, o_2\}$. Let $S_1 = \langle (f_4, s_4), (f_5, s_5) \rangle$, $S_2 = \langle (f_3, s_3), (f_4, s_4) \rangle$, $S_3 = \langle (f_2, s_2), (f_3, s_3) \rangle$ be sequences occurring in v. S_1 is totally (and partially) unexplained in every world. S_2 is totally unexplained in w_0 and w_2 but not in w_1 and w_3; moreover, S_2 is partially unexplained in every world. S_3 is totally and partially unexplained in w_0 and w_2 but not in w_1 and w_3.

We now define the probability of a sequence in an observation sequence being totally/partially unexplained.

Definition 3.2.4 *Let S be a sequence occurring in an observation sequence v. The probability interval that S is totally unexplained in v is $\mathcal{I}_T(S) = [l, u]$, where:*

$$l = \textbf{minimize} \sum_{w \in \mathcal{W} \text{ s.t. } w \not\models_T S} p_w \quad \textbf{subject to } NLC(v)$$

$$u = \textbf{maximize} \sum_{w \in \mathcal{W} \text{ s.t. } w \not\models_T S} p_w \quad \textbf{subject to } NLC(v)$$

The probability interval that S is partially unexplained in v is $\mathcal{I}_P(S) = [l', u']$, where l', u' are derived in exactly the same way as l, u above by replacing the $\not\models_T$ symbols in the above optimization problems by $\not\models_P$.

Thus, the probability that a sequence S occurring in v is totally (resp. partially) unexplained w.r.t. a solution of $NLC(v)$ is the sum of the probabilities of the worlds in which S is totally (resp. partially) unexplained. As $NLC(v)$ may have multiple solutions, we find the tightest interval $[l, u]$ (resp. $[l', u']$) containing this probability for any solution. Different criteria can be used to infer a value from an interval $[l, u]$, e.g. the MIN l, the MAX u, the average (i.e., $(l + u)/2$), etc. The only requirement is that this value has to be in $[l, u]$. We henceforth assume that such criterion has been chosen—$\mathcal{P}_T(S)$ (resp. $\mathcal{P}_P(S)$) denotes the probability that S is totally (resp. partially) unexplained.

The following proposition says that the probability that a sequence is totally (resp. partially) unexplained is no higher (resp. lower) than the probability of any subsequence.

Proposition 3.2.1 *Consider two sequences S_1 and S_2 occurring in an observation sequence. If S_1 is a subsequence of S_2, then $\mathcal{P}_T(S_1) \geq \mathcal{P}_T(S_2)$ and $\mathcal{P}_P(S_1) \leq \mathcal{P}_P(S_2)$.*

We now define totally and partially unexplained sequences.

Definition 3.2.5 (Unexplained sequences) *Let v be an observation sequence, $\tau \in [0, 1]$ a probability threshold, and $L \in \mathbb{N}^+$ a length threshold. A sequence S occuring in v is:*

- *A* totally unexplained sequence *if (i) $\mathcal{P}_T(S) \geq \tau$, (ii) $|S| \geq L$, and (iii) S is maximal, i.e., there is no sequence $S' \neq S$ occurring in v s.t. S is a subsequence of S', $\mathcal{P}_T(S') \geq \tau$, and $|S'| \geq L$.*
- *A* partially unexplained sequence *if (i) $\mathcal{P}_P(S) \geq \tau$, (ii) $|S| \geq L$, and (iii) S is minimal, i.e., there is no sequence $S' \neq S$ occurring in v s.t. S' is a subsequence of S, $\mathcal{P}_P(S') \geq \tau$, and $|S'| \geq L$.*

In this definition, L is the minimum length a sequence must be for it to be considered possibly unexplained. Totally unexplained sequences S have to be maximal because once we find S, any sub-sequence of it is (totally) unexplained with probability greater than or equal to that of S. On the other hand, partially unexplained sequences S' have to be minimal because once we find S', any super-sequence of it is (partially) unexplained with probability greater than or equal to that of S'.

Intuitively, an unexplained sequence is a sequence of action symbols that are observed in the observation sequence and poorly explained by known activity models. Such sequences might correspond to unknown variants of known activities or to entirely new—and unknown—activities. So, in this way, we can discover new activities not sufficiently explained by the well-known models and that can be further analyzed by final users.

An *Unexplained Activity Problem* (UAP) instance is a triple $I = \langle v, \tau, L \rangle$ where v is an observation sequence, $\tau \in [0, 1]$ is a probability threshold, and $L \in \mathbb{N}^+$ is a length threshold. We want to find the sets $\mathcal{A}^{tu}(I)$ and $\mathcal{A}^{pu}(I)$ of all totally and partially unexplained activities, respectively. When I is clear from context, we will drop it.

The following definition introduces the top-k totally and partially unexplained activities. Intuitively, these are k unexplained activities having maximum probability.

Definition 3.2.6 (Top-k unexplained activities) *Consider a UAP instance and let $k \in \mathbb{N}^+$. $\mathcal{A}^{tu}_k \subseteq \mathcal{A}^{tu}$ (resp. $\mathcal{A}^{pu}_k \subseteq \mathcal{A}^{pu}$) is a set of top-k totally (resp. partially) unexplained activities iff $|\mathcal{A}^{tu}_k| = \min\{k, |\mathcal{A}^{tu}|\}$ (resp. $|\mathcal{A}^{pu}_k| = \min\{k, |\mathcal{A}^{pu}|\}$), and $\forall S \in \mathcal{A}^{tu}_k, \forall S' \in \mathcal{A}^{tu} - \mathcal{A}^{tu}_k$ (resp. $\forall S \in \mathcal{A}^{pu}_k, \forall S' \in \mathcal{A}^{pu} - \mathcal{A}^{pu}_k$) $\mathcal{P}_T(S) \geq \mathcal{P}_T(S')$ (resp. $\mathcal{P}_P(S) \geq \mathcal{P}_P(S')$).*

Suppose we have a UAP instance. For any $S, S' \in \mathcal{A}^{tu}$ (resp. $S, S' \in \mathcal{A}^{pu}$), we write $S =_T S'$ (resp. $S =_P S'$) iff $\mathcal{P}_T(S) = \mathcal{P}_T(S')$ (resp. $\mathcal{P}_P(S) = \mathcal{P}_P(S')$). Obviously, $=_T$ (resp. $=_P$) is an equivalence relation and determines a set $\mathcal{C}^{tu}$ (resp. $\mathcal{C}^{pu}$) of equivalence classes. For any equivalence class $C \in \mathcal{C}^{tu}$ (resp. $C \in \mathcal{C}^{pu}$) we define $\mathcal{P}_T(C)$ (resp. $\mathcal{P}_P(C)$) as the (unique) probability of the sequences in C.

Compared with the top-k unexplained activities, the top$-k$ unexplained classes find *all* the unexplained sequences having the k highest probabilities.

Definition 3.2.7 (Top-k unexplained classes) *Consider a UAP instance and let $k \in \mathbb{N}^+$. $\mathcal{C}^{tu}_k \subseteq \mathcal{C}^{tu}$ (resp. $\mathcal{C}^{pu}_k \subseteq \mathcal{C}^{pu}$) is the set of top-k totally (resp. partially) unexplained classes iff $|\mathcal{C}^{tu}_k| = \min\{k, |\mathcal{C}^{tu}|\}$ (resp. $|\mathcal{C}^{pu}_k| = \min\{k, |\mathcal{C}^{pu}|\}$), and $\forall C \in \mathcal{C}^{tu}_k, \forall C' \in \mathcal{C}^{tu} - \mathcal{C}^{tu}_k$ (resp. $\forall C \in \mathcal{C}^{pu}_k, \forall C' \in \mathcal{C}^{pu} - \mathcal{C}^{pu}_k$) $\mathcal{P}_T(C) > \mathcal{P}_T(C')$ (resp. $\mathcal{P}_P(C) > \mathcal{P}_P(C')$).*

Table 3.1 summarizes the main notation used in the thesis.

Symbol	Description
$\mathcal{A}$	Set of stochastic activities
s	Action symbol
f	Observation ID (OID)
$f.ts$	Timestamp associated with observation ID f
$f.obs$	Set of action symbols associated with observation ID f
v	Observation sequence
o and $\mathcal{O}$	Activity occurrence and set of activity occurrences
w and $\mathcal{W}$	Possible world and set of possible worlds
$\langle v_1, \ldots, v_m \rangle$	Conflict-based partitioning (CBP) of observation sequence v. Each v_i is called a *segment*
$NLC(v)$	Set of non-linear constraints for observation sequence v
$LC(v)$	Set of linear constraints for observation sequence v
$w \not\models_T S$	Sequence S is totally unexplained in world w
$w \not\models_P S$	Sequence S is partially unexplained in world w
$\mathcal{I}_T(S)$	Probability interval that sequence S is totally unexplained
$\mathcal{I}_P(S)$	Probability interval that sequence S is partially unexplained
$\mathcal{P}_T(S)$	(Point) Probability that sequence S is totally unexplained
$\mathcal{P}_P(S)$	(Point) Probability that sequence S is partially unexplained

Table 3.1: Notation

3.2.1 Properties

This subsection derives properties that can be leveraged (in chapter 4) to devise efficient algorithms to solve UAPs. We first show an interesting property concerning the solution of $NLC(v)$ (some later results rely on it); the following two sections consider specific properties for totally and partially unexplained activities.

For a given observation sequence v, we show that if $\langle v_1, \ldots, v_m \rangle$ is a CBP, then we can find solutions of the *non-linear* constraints $NLC(v)$ by solving m *smaller sets of linear constraints.*

This yields some benefits:

- It allows us to solve a smaller set of constraints.
- It allows us to solve linear constraints which are easier to solve than nonlinear ones.
- Moreover, it allows us to drastically reduce the space of possible worlds considered, as we can consider each segment v_i (and its corresponding possible worlds)

individually, thereby avoiding the blow up we would get by combining possible worlds of different segments. This also applies to Theorems 2 and 4.

Let $LC(v)$ be the set of linear constraints of $NLC(v)$ (i.e., all constraints of Definition 3.2.3 except for the last kind). Henceforth, we use $\mathcal{W}$ to denote $\mathcal{W}(v)$ and $\mathcal{W}_i$ to denote $\mathcal{W}(v_i)$, $1 \leq i \leq m$. A solution of $NLC(v)$ is a mapping $\mathcal{P} : \mathcal{W} \to [0,1]$ which satisfies $NLC(v)$. Likewise, a solution of $LC(v_i)$ is a mapping $\mathcal{P}_i : \mathcal{W}_i \to [0,1]$ which satisfies $LC(v_i)$. It is important to note that $\mathcal{W} = \{w_1 \cup \cdots \cup w_m \mid w_i \in \mathcal{W}_i, 1 \leq i \leq m\}$.

Theorem 1 *Let v be an observation sequence and $\langle v_1, \ldots, v_m \rangle$ a CBP. $\mathcal{P}$ is a solution of $NLC(v)$ iff $\forall i \in [1,m]$ there exists a solution $\mathcal{P}_i$ of $LC(v_i)$ s.t. $\mathcal{P}(\bigcup_{i=1}^{m} w_i) = \prod_{i=1}^{m} \mathcal{P}_i(w_i)$ for every $w_1 \in \mathcal{W}_1, \ldots, w_m \in \mathcal{W}_m$.*

The following example illustrates the previous theorem.

Example 3.2.7 (Video example) *Consider the video v of Example 3.2.3 (cf. Figure 3.2). As shown in Example 3.2.4, one possible CBP of v is $\langle v_1, v_2 \rangle$, where $v_1 = \langle f_1, \ldots, f_9 \rangle$ and $v_2 = \langle f_{10}, \ldots, f_{16} \rangle$. Theorem 1 says that for each solution $\mathcal{P}$ of $NLC(v)$, there is a solution $\mathcal{P}_1$ of $LC(v_1)$ and a solution $\mathcal{P}_2$ of $LC(v_2)$ s.t. $\mathcal{P}(w_1 \cup w_2) = \mathcal{P}_1(w_1) \times \mathcal{P}(w_2)$ for every $w_1 \in \mathcal{W}_1, w_2 \in \mathcal{W}_2$, and vice versa.*

Consider an observation sequence v and let $\langle v_1, \ldots, v_m \rangle$ be a CBP. Given a sequence $S = \langle (f_1, s_1), \ldots, (f_q, s_q) \rangle$ occurring in v, we say that $v_i, v_{i+1}, \ldots, v_{i+n}$ ($1 \leq i \leq i+n \leq m$) are the segments *containing* S iff $f_1 \in v_i$ and $f_q \in v_{i+n}$. In other words, S spans the segments $v_i, v_{i+1}, \ldots, v_{i+n}$: it starts at a point in segment v_i (as v_i contains the first OID of S) and ends at some point in segment v_{i+n} (as v_{i+n} contains the last OID of S). S_k denotes the *projection* of S on the k-th segment v_k ($i \leq k \leq i+n$), that is, the subsequence of S containing all the pairs $(f, s) \in S$ with $f \in v_k$.

Example 3.2.8 (Video example) *Suppose we have a video $v = \langle f_1, \ldots, f_{21} \rangle$ such that $f_i.obs = \{s_i\}$ for $1 \leq i \leq 21$. In addition, suppose 8 occurrences are detected as shown in Figure 3.4. Consider the CBP $\langle v_1, v_2, v_3, v_4 \rangle$, where $v_1 = \{f_1, \ldots, f_5\}$, $v_2 = \{f_6, \ldots, f_{10}\}$, $v_3 = \{f_{11}, \ldots, f_{16}\}$, and $v_4 = \{f_{17}, \ldots, f_{21}\}$. Consider now the sequence $S = \langle (f_8, s_8), \ldots, (f_{14}, s_{14}) \rangle$ occurring in v. Then, v_2 and v_3 are the segments containing S. Moreover, S_2 denotes $\langle (f_8, s_8), \ldots, (f_{10}, s_{10}) \rangle$, and S_3 denotes $\langle (f_{11}, s_{11}), \ldots, (f_{14}, s_{14}) \rangle$.*

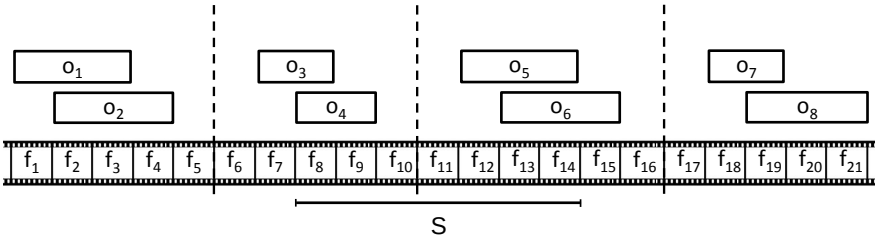


Figure 3.4: Conflict-Based Partitioning of a video

3.2.1.1 Totally Unexplained Activities

The following theorem says that we can compute $\mathcal{I}_T(S)$ by solving LC (which are linear constraints) for each segment containing S (instead of solving a non-linear set of constraints for the whole observation sequence).

Theorem 2 *Consider an observation sequence v. Let $\langle v_1, \ldots, v_m \rangle$ be a CBP and $\langle v_i, \ldots, v_{i+n} \rangle$ the segments containing a sequence S occurring in v. For $i \leq k \leq i+n$, let*

$$l_k = \mathbf{minimize} \sum_{w \in \mathcal{W}_k \ s.t. \ w \not\models_T S_k} p_w \quad \mathbf{subject\ to}\ LC(v_k)$$

$$u_k = \mathbf{maximize} \sum_{w \in \mathcal{W}_k \ s.t. \ w \not\models_T S_k} p_w \quad \mathbf{subject\ to}\ LC(v_k)$$

If $\mathcal{I}_T(S) = [l, u]$, then $l = \prod_{k=i}^{i+n} l_k$ and $u = \prod_{k=i}^{i+n} u_k$.

The following example illustrates the theorem above.

Example 3.2.9 (Video example) *Consider Example 3.2.8, which is depicted in Figure 3.4. $\mathcal{I}_T(S)$ can be computed by solving the non-linear program of Definition 3.2.4 for the whole video v. But Theorem 2 says that $\mathcal{I}_T(S)$ can be computed as $\mathcal{I}_T(S) = [l_2 \times l_3, u_2 \times u_3]$, where l_2, u_2, l_3, u_3 are computed as defined in Theorem 2, i.e. by solving two smaller linear programs for v_2 and v_3.*

The following theorem provides a sufficient condition for a pair (f, s) not to be included in any sequence S occurring in v and having $\mathcal{P}_T(S) \geq \tau$.

Theorem 3 *Let $\langle v, \tau, L\rangle$ be a UAP instance. Given (f, s) s.t. $f \in v$ and $s \in f.obs$, let $\varepsilon = \sum_{o \in \mathcal{O}\ s.t.\ (f,s)\in o} p^*(o) \cdot \frac{Weight(o)}{\sum_{o_j \in C(o)} Weight(o_j)}$. If $\varepsilon > 1 - \tau$, then there does not exist a sequence S occurring in v s.t. $(f, s) \in S$ and $\mathcal{P}_T(S) \geq \tau$.*

If the above condition holds for a pair (f, s), then we say that (f, s) is *sufficiently explained. Note that to check whether a pair (f, s) is sufficiently explained, we do not need to solve any set of linear or non-linear constraints, since ε is computed by simply summing the (weighted) probabilities of the occurrences containing (f, s).* Thus, this result yields a further efficiency. An OID f is *sufficiently explained* iff (f, s) is sufficiently explained for every $s \in f.obs$. If (f, s) is sufficiently explained, then it can be disregarded when identifying unexplained sequences. Moreover, this may allow us to disregard entire parts of observation sequences as shown in the example below.

Example 3.2.10 (Video example) *Consider a UAP instance $\langle v, \tau, L\rangle$ where $v = \langle f_1, \ldots, f_9\rangle$ is s.t. $f_i.obs = \{s_i\}$ for $1 \leq i \leq 9$, as depicted in Figure 3.5.*

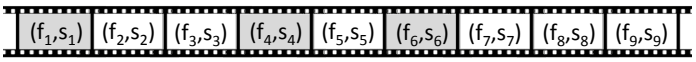


Figure 3.5: Sufficiently explained frames in a video

Suppose $L = 3$ and (f_1, s_1), (f_4, s_4), (f_6, s_6) are sufficiently explained. Even though the theorem is applicable to only a few (f_i, s_i) pairs, we see that no unexplained sequence can be found before f_7 as $L = 3$.

Given a UAP instance $I = \langle v, \tau, L\rangle$ and a subsequence v' of v, v' is *relevant* iff (i) v' is a contiguous subsequence of v (ii) $|v'| \geq L$, (iii) $\forall f \in v'$, f is not sufficiently explained, and (iv) v' is maximal (i.e., there does not exist $v'' \neq v'$ s.t. v' is a subsequence of v'' and v'' satisfies (i), (ii), (iii)). We use $relevant(I)$ to denote the set of relevant observation subsequences.

Theorem 3 entails that relevant observation subsequences can be individually considered when looking for totally unexplained sequences because there is no totally unexplained sequence spanning two different relevant observation subsequences.

3.2.1.2 Partially Unexplained Activities

The following theorem states that we can compute $\mathcal{I}_P(S)$ by solving NLC for the observation subsequence consisting of the segments containing S (instead of solving NLC for the whole observation sequence).

Theorem 4 *Consider an observation sequence v. Let $\langle v_1, \ldots, v_m \rangle$ be a CBP and $\langle v_i, \ldots, v_{i+n} \rangle$ be the segments containing a sequence S occurring in v. Let $v^* = v_i \cdot \cdots \cdot v_{i+n}$. $\mathcal{I}_P(S)$ computed w.r.t. v is equal to $\mathcal{I}_P(S)$ computed w.r.t. v^*.*

We now illustrate the use of the preceding theorem.

Example 3.2.11 (Video example) *Consider Example 3.2.8 as shown in Figure 3.4. By definition, $\mathcal{I}_P(S)$ can be computed by solving the non-linear program of Definition 3.2.4 for the whole video v. Alternatively, Theorem 4 says that $\mathcal{I}_P(S)$ can be computed by solving the non-linear program of Definition 3.2.4 for the sub-video $v^* = v_2 \cdot v_3$.*

Chapter 4

Unexplained Activity Detection Algorithms

We now present algorithms to find top-k totally and partially unexplained activities and classes. For ease of presentation, we assume $|f.obs| = 1$ for every OID f in an observation sequence (this makes the algorithms much more concise – generalization to the case of multiple action symbols per OID is straightforward). It suffices to consider the different sequences given by the different action symbols. Given an observation sequence $v = \langle f_1, \ldots, f_n \rangle$, we use $v(i, j)$ $(1 \leq i \leq j \leq n)$ to denote the sequence $S = \langle (f_i, s_i), \ldots, (f_j, s_j) \rangle$, where s_k is the only element in $f_k.obs$, $i \leq k \leq j$.

4.1 Top-k TUA and TUC

The Top-k TUA algorithm computes a set of top-k totally unexplained activities in an observation sequence. Note that:

- at every time, $lowest$ is defined as follows:

$$lowest = \begin{cases} -1 & \text{if } |TopSol| < k \\ \min\{\mathcal{P}_T(S) \mid S \in TopSol\} & \text{if } |TopSol| = k \end{cases}$$

- On line 30, "Add S to $TopSol$" works as follows:
  - If $|TopSol| < k$, then S is added to $TopSol$;

– otherwise, a sequence S' in $TopSol$ having minimum $\mathcal{P}_T(S')$ is replaced by S.

Algorithm 1 Top-k TUA

Input: UAP instance $I = \langle v, \tau, L \rangle$, $k \geq 1$
Output: Top-k totally unexplained activities

```
 1: TopSol = ∅
 2: for all v' ∈ relevant(I) do
 3:     start = 1; end = L
 4:     repeat
 5:         if P_T(v'(start, end)) ≥ τ ∧ P_T(v'(start, end)) > lowest then
 6:             end' = end
 7:             while end < |v'| do
 8:                 end = min{end + L, |v'|}
 9:                 if P_T(v'(start, end)) < τ then
10:                     break
11:                 else
12:                     if P_T(v'(start, end)) ≤ lowest then
13:                         end = end + 1
14:                         go to line 33
15:             s = max{end − L, end'}; e = end
16:             while e ≠ s do
17:                 mid = ⌈(s + e)/2⌉
18:                 if P_T(v'(start, mid)) ≥ τ then
19:                     if P_T(v'(start, mid)) ≤ lowest then
20:                         end = mid + 1
21:                         go to line 33
22:                     else
23:                         s = mid
24:                 else
25:                     e = mid − 1
26:             if start > 1 ∧ P_T(v'(start − 1, s)) ≥ τ then
27:                 end = s + 1
28:                 go to line 33
29:             else
30:                 S = v'(start, s); Add S to TopSol
31:                 start = start + 1; end = s + 1
32:         else
33:             start = start + 1;  end = max{end, start + L − 1}
34:     until end > |v'|
35: return TopSol
```

Leveraging Theorem 3, Top-k TUA considers only relevant observation subsequences of v individually (line 2). When it finds a sequence $v'(start, end)$ of length at least L having a probability of being totally unexplained greater than $lowest$ (line 5), it makes the sequence maximal by adding OIDs on the right (lines 7–14). Instead of adding one OID at a time, $v'(start, end)$ is extended by L OIDs at a time until its probability drops below τ (lines 9–10); a binary search is then performed to find the exact maximum length of the unexplained sequence (lines 15–25). While making the sequence maximal, if the

algorithm realizes that the unexplained sequence will not have a probability greater than $lowest$ (i.e., the sequence is not a top-k TUA), then the sequence is disregarded and the process of making the sequence maximal is aborted (lines 12–14 and 19–21). This pruning allows the algorithm to move forward in the observation sequence avoiding computing the exact ending OID of the TUA thereby saving time. Throughout the algorithm, $\mathcal{P}_T$ is computed by applying Theorem 2.

Theorem 5 *Algorithm Top-k TUA returns a set of top-k totally unexplained activities of the input instance.*

Algorithm Top-k TUC modifies Top-k TUA as follows to compute the top-k totally unexplained classes:

- At every time, $lowest$ is defined as follows:

$$lowest = \begin{cases} -1 & \text{if } |TopSol| < k \\ \min\{\mathcal{P}_T(C) \mid C \in TopSol\} & \text{if } |TopSol| = k \end{cases}$$

- "Add S to $TopSol$" (line 30) works as follows:
  - If there exists $C \in TopSol$ s.t. $\mathcal{P}_T(C) = \mathcal{P}_T(S)$, then S is added to C;
  - else if $|TopSol| < k$, then the class $\{S\}$ is added to $TopSol$;
  - otherwise the class C in $TopSol$ having minimum $\mathcal{P}_T(C)$ is replaced with $\{S\}$.
- On line 5, $\mathcal{P}_T(v'(start, end)) > lowest$ is replaced with $\mathcal{P}_T(v'(start, end)) \geq lowest$;
- On line 12, $\mathcal{P}_T(v'(start, end)) \leq lowest$ is replaced with $\mathcal{P}_T(v'(start, end)) < lowest$;
- On line 19, $\mathcal{P}_T(v'(start, mid)) \leq lowest$ is replaced with $\mathcal{P}_T(v'(start, mid)) < lowest$;

The algorithm obtained by applying the modifications above is named Top-k TUC.

Theorem 6 *Algorithm Top-k TUC returns the top-k totally unexplained classes of the input instance.*

4.2 Top-k PUA and PUC

The Top-k PUA algorithm below computes a set of top-k partially unexplained activities in an observation sequence. Note that:

- at each time, $lowest$ is defined as follows:

$$lowest = \begin{cases} -1 & \text{if } |TopSol| < k \\ \min\{\mathcal{P}_P(S) \mid S \in TopSol\} & \text{if } |TopSol| = k \end{cases}$$

- On line 43, "Add S to $TopSol$" works as follows:
  - If $|TopSol| < k$, then S is added to $TopSol$;
  - otherwise, a sequence in $TopSol$ having minimum $\mathcal{P}_P$ is replaced by S.

To find an unexplained activity, Algorithm Top-k PUA starts with a sequence of length at least L and adds OIDs to its right until its probability of being partially unexplained is above the threshold. As in the case of Top-k TUA, this is done by adding L OIDs at a time (lines 5–8) and then performing a binary search (lines 9–27). When performing the binary search, if at some point the algorithm realizes that the partially unexplained sequence will not have a probability greater than $lowest$, then the sequence is disregarded and the binary search is aborted (lines 17–19 and lines 24–25). Otherwise, the sequence is shortened on the left making it minimal (lines 28–38) by performing a binary search instead of proceeding one OID at a time. If the algorithm realizes that the partially unexplained sequence will not have a probability greater than $lowest$, then the sequence is disregarded and the shortening process is aborted (lines 34–36). This allows the algorithm to avoid computing the exact starting OID of the PUS, thus saving time. Note that $\mathcal{P}_P$ is computed by applying Theorem 4.

Theorem 7 *Algorithm Top-k PUA returns the set of top-k partially unexplained activities of the input instance.*

Algorithm Top-k PUC modifies Top-k PUA as follows to compute the top-k partially unexplained classes:

Algorithm 2 Top-k PUA

Input: UAP instance $I = \langle v, \tau, L \rangle$, $k \geq 1$
Output: Top-k partially unexplained activities

1: $TopSol = \emptyset$; $start = 1$; $end = L$
2: **while** $end \leq |v|$ **do**
3: **if** $\mathcal{P}_P(v(start, end)) < \tau$ **then**
4: $end' = end$
5: **while** $end < |v|$ **do**
6: $end = \min\{end + L, |v|\}$
7: **if** $\mathcal{P}_P(v(start, end)) \geq \tau$ **then**
8: **break**
9: **if** $\mathcal{P}_P(v(start, end)) \geq \tau$ **then**
10: **if** $\mathcal{P}_P(v(start, end)) > lowest$ **then**
11: $s = \max\{end' + 1, end - L + 1\}$; $e = end$
12: **while** $e \neq s$ **do**
13: $mid = \lfloor (s + e)/2 \rfloor$
14: **if** $\mathcal{P}_P(v(start, mid)) < \tau$ **then**
15: $s = mid + 1$
16: **else**
17: **if** $\mathcal{P}_P(v(start, mid)) \leq lowest$ **then**
18: $start = start + 1$; $end = mid + 1$
19: **go to** line 2
20: **else**
21: $e = mid$
22: $end = e$
23: **else**
24: $start = start + 1$; $end = end + 1$
25: **go to** line 2
26: **else**
27: **return** $TopSol$
28: $s' = start$; $e' = end - L + 1$
29: **while** $e' \neq s'$ **do**
30: $mid = \lceil (s' + e')/2 \rceil$
31: **if** $\mathcal{P}_P(v(mid, end)) < \tau$ **then**
32: $e' = mid - 1$
33: **else**
34: **if** $\mathcal{P}_P(v(mid, end)) \leq lowest$ **then**
35: $start = mid + 1$; $end = end + 1$
36: **go to** line 2
37: **else**
38: $s' = mid$
39: **if** $\mathcal{P}_P(v(s', end - 1)) \geq \tau \wedge |v(s', end - 1)| \geq L$ **then**
40: $start = s' + 1$; $end = end + 1$
41: **go to** line 2
42: **else**
43: $S = v(s', end)$; Add S to $TopSol$
44: $start = s' + 1$; $end = end + 1$
45: **return** $TopSol$

- At every time, $lowest$ is defined as follows:

$$lowest = \begin{cases} -1 & \text{if } |TopSol| < k \\ \min\{\mathcal{P}_P(C) \mid C \in TopSol\} & \text{if } |TopSol| = k \end{cases}$$

- "Add S to $TopSol$" (line 43) works as follows:
  - If there exists $C \in TopSol$ s.t. $\mathcal{P}_P(C) = \mathcal{P}_P(S)$, then S is added to C;
  - else if $|TopSol| < k$, then the class $\{S\}$ is added to $TopSol$;
  - otherwise the class C in $TopSol$ having minimum $\mathcal{P}_P(C)$ is replaced with $\{S\}$.
- On line 10, $\mathcal{P}_P(v(start, end)) > lowest$ is replaced with $\mathcal{P}_P(v(start, end)) \geq lowest$;
- On line 17, $\mathcal{P}_P(v(start, mid)) \leq lowest$ is replaced with $\mathcal{P}_P(v(start, mid)) < lowest$;
- On line 34, $\mathcal{P}_P(v(mid, end)) \leq lowest$ is replaced with $\mathcal{P}_P(v(mid, end)) < lowest$;

The algorithm obtained by applying the modifications above is named Top-k PUC.

Theorem 8 *Algorithm Top-k PUC returns the top-k partially unexplained classes of the input instance.*

Chapter 5

Experimental evaluation

We implemented Algorithms Top-k TUA, Top-k PUA, Top-k TUC and Top-k PUC, and experimentally evaluated both running time and accuracy on real-world video and cyber security datasets.

5.1 Video analysis

We note that identifying frame observations via the development of image processing algorithms is an extremely challenging task—the goal of our approach is to present a domain-independent way of identifying unexplained activities that builds upon domain-specific ways of recognizing actions in observation sequences. Anyway, in order to address this issue and to make a complete experimentation on real-world datasets, we have designed and developed a specific prototype implementation for video surveillance domain, which is able to automatically catch *observation IDs* with a reasonable accuracy and to go on with the analysis until the unexplained activities are discovered. Such a prototype is described in detail in 5.1.1 subsection.

5.1.1 The developed prototype

As we can see in figure 5.1, our prototype consists of:

- an *Image Processing Library*
- a *Video Labeler*

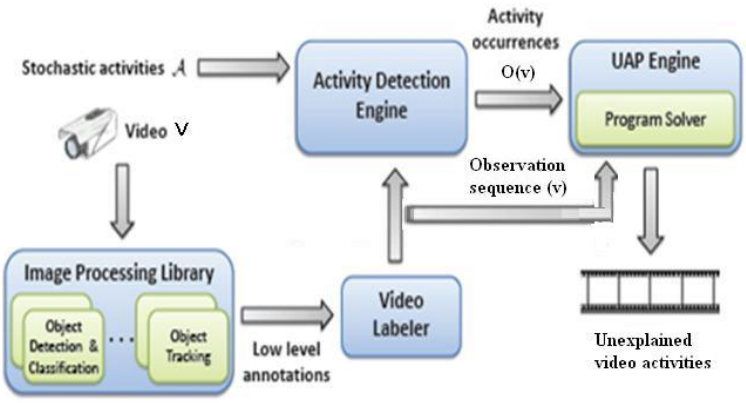


Figure 5.1: The prototype architecture for video context

- an *Activity Detection Engine*
- the *UAP Engine* implementing the algorithms described in chapter 4

In particular, the *Image Processing Library* analyzes the video captured by sensors / cameras and returns the low level annotations for each video frame as output; the *Video Labeler* fills the semantic gap between the low level annotations captured for each frame and high level annotations, representing the observation sequence as defined in chapter 3; then, we used an *Activity Detection Engine* to find activity occurrences matching the well-known models, that can be classified into *good* and *bad* ones, as defined in section 1.2: thus, such a module takes as inputs the *observation sequence* previously caught by the *Video Labeler* and the stochastic activity models; finally, our framework, called *UAP Engine* in this overall architecture, takes as input the activity occurrences previously found with the associated probabilities and the observation sequence and discovers the *Unexplained Video Activities*.

All these components will be described into subsubsections 5.1.1.1, 5.1.1.2, 5.1.1.3, 5.1.1.4.

5.1.1.1 The Image Processing Library

With recent advances of computer technology automated visual surveillance has become a popular area for research and development. Surveillance cameras are installed in many

public areas to improve safety, and computer-based image processing is a promising means to handle the vast amount of image data generated by large networks of cameras. A number of algorithms to track people in camera images can be found in the literature, yet so far little research has gone into building realtime visual surveillance systems that integrate people trackers. The task of an integrated surveillance system is to warn an operator when it detects events which may require human intervention, for example to avoid possible accidents or vandalism. These warnings can only be reliable if the system can detect and understand human behaviour, and for this it must locate and track people reliably. Thus, a moltitude of *Computer Vision* algorithms have been developed.

Unfortunately, another issue of this kind of algorithms is due to false alarms and lost events, which can reduce the system reliability; such a reliability is strongly related to the environmental conditions in which the system works. Basically, the contribution of *Computer Vision* on this kind of applications is based on *difference analysis* with a more or less well-known model of reality. For instance, for what concerns the pixel identification of moving objects, there is made a subtraction between the current image and the pixels of a background model. Either, when objects are tracked, the object shapes are compared with some reference models. A main aspect to be taken into account is how much the environmental variations dissociate from the model. Such variations can be in some cases due to the noise. While in indoor environments the presence of noises is quite limited, in outdoor environments such a noise may be unrestrained and excessive and can cause a considerable degradation of quality performances. There are numerous applications used for these purposes. Anyway, this kind of processing can be structured into different layers:

- *pixel layer*
- *frame layer*
- *tracking layer*

At the lowest layer there is located the *pixel processing layer*, which processes the images captured from the source in a strictly related to pixel domain. Through the base algorithms, a pixel identification is made in order to verify if pixels belong either to objects or to the background of the scene. The proposed video surveillance applications have used different kinds of *pixel processing* algorithms in the years. Basically, the pixel

identification can be made either through *background difference* techniques or through *frame difference* ones or through combinations of both of them. The knowledge at this layer is limited at the image pixels: so, the nature and the number of the present objects are not considered. Moreover, noise sources have to be taken into account. As a matter of fact, a specific tuning phase is needed to restrict the presence of pixels caused by noise.

In the second processing layer (*Frame Processing Layer*), a first interpretation of the image supplied by the underlying layer is executed. The relationships between pixels detected as zone of interest are considered. The techiniques used in this context tend to cluster the pixel found at the underlying layer, trying to erase pixels not belonging to the interested objects and representing noise (through *size filtering* techniques) and other ones which, though representing movement, belong to not-interested objects, like shadows or areas due to lighting effects. The main aim of such a layer is to cluster the pixels belonging to the interested objects (blobs) through different segmentation techniques. Moreover, another *Frame Processing* task consists of the feature extraction of the detected objects. This information will be the base for the next layer.

The *Tracking layer* aim consists of the pursuit and classification of the objects discovered at the previous layer. It needs the knowledge of some information about the objects previously found. For instance, this information can usually deal with size, area, shape and all the properties that may help to identify a pixel aggregate present into different and often consecutive frames of the image sequence as an only object. For what concerns video surveillance systems, there are many aspects to be taken into account for an effective implementation. Depending on the particular application domain, there are many constraints that may cause strong consequences on the whole process. Depending on the application type, it is possible to identify some restricted zone of the image (*Region of Interest*), where the overall attention on the video surveillance process is focused. There are some specific applications for *indoor* and *outdoor* environments and other ones specialized in detections of particular classes of objects (as persons, cars and so on...).

The Image Processing Library used in our prototype implementation is the *Reading People Tracker (RPT)* [52], [53]. This library achieved a good accuracy in object detection and tracking and is very easy to be installed and used on Unix systems. Moreover, it returns a *XML-based* output format, that is very easy to understand and process.

More in details, the Reading People Tracker is a software for tracking people in camera images for visual surveillance purposes. It originates from research work on people

tracking for automatic visual surveillance systems for crime detection and prevention. It was built within the context of two PhD theses (by AM Baumberg and NT Siebel) and contains state-of-the art image processing algorithms. It is easily maintainable and well documented. Therefore it can (and has already been) easily be adapted to new requirements and different projects. The Reading People Tracker contains the necessary functionality to read video sequences from hard disk or a video camera (IEEE1394/DV), to manipulate the images with image filters and to analyse them with a number of detection and tracking modules. The Reading People Tracker as it exists today was developed by *Nils T Siebel* in the European Framework V research project *ADVISOR*. It is based on the *Leeds People Tracker* which was developed by *Adam Baumberg*. Starting from there it took 3 years of work on software and algorithms to develop what is now called the "Reading People Tracker". Now that the ADVISOR project has finished the Reading People Tracker is still being maintained and available for download. Recent code changes include better support for newer compilers and a number of bugfixes. There has also been some support by the community in the form of bugfixes and a few small additions. This has increased stability and ease of use. The tracking functionality itself has not changed significantly since release 1.25 in 2003. However, an update to the newest version is strongly recommended for all current users.

Thus, in order for an automated visual surveillance system to operate effectively it must locate and track people reliably and in realtime. The Reading People Tracker achieves this by combining four detection and tracking modules, each of medium to low complexity. As we said before, the Reading People Tracker can work either standalone or as a subsystem of the ADVISOR system. The focus here is on tracking, specifically on how a number of detection and tracking algorithms can be combined to achieve robust tracking of people in an indoor environment.

Automated visual surveillance systems have to operate in realtime and with a minimum of hardware requirements, if the system is to be economical and scalable. This limits the complexity of models that can be used for detection and tracking. Any attempt at designing a People Tracker for a surveillance system like *ADVISOR* therefore has to consider the realtime aspect during algorithm design.

Figure 5.2 shows the overall system layout, with individual subsystems for tracking, detection and analysis of events, together with storage and human-computer interface subsystems to meet the needs of the surveillance system operators. Each of these sub-

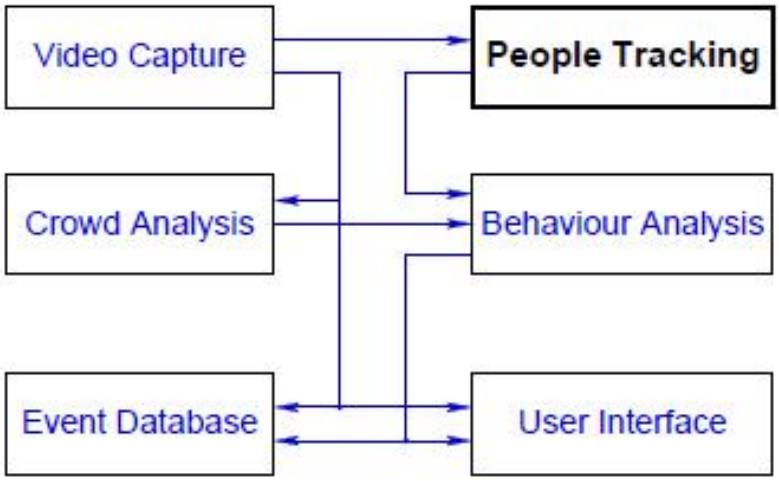


Figure 5.2: People Tracking as one of six subsystems of ADVISOR

systems is designed to run in realtime on off-the-shelf PC hardware, with the ability to process video input from a number of cameras simultaneously. The connections between the subsystems are realised by Ethernet. Images are transferred across the network using *JPEG* image compression. Other data, such as the output of the People Tracker and the results of the Behaviour Analysis, are represented in XML formats defined by a number of *XML Schemas*.

The Reading People Tracker has been designed to run either as a subsystem of ADVISOR or in standalone mode. For its design four detection and tracking modules of medium to low complexity have been chosen, improved and combined in a single tracking system.

Originally based on the *Leeds People Tracker*, the most important one of the four modules is a slightly modified version of Adam Baumbergs Active Shape Tracker. The people tracker has been modified over time to increase tracking robustness, and adapted for use in *ADVISOR*. The tracker was ported from a *SGI* platform to a PC running *GNU/Linux* to facilitate economical system integration.

The People Tracking subsystem is itself comprised of four modules which cooperate to create the overall tracking output, aiding each other to increase tracking robustness and to overcome the limitations of individual modules. The following passages will focus on

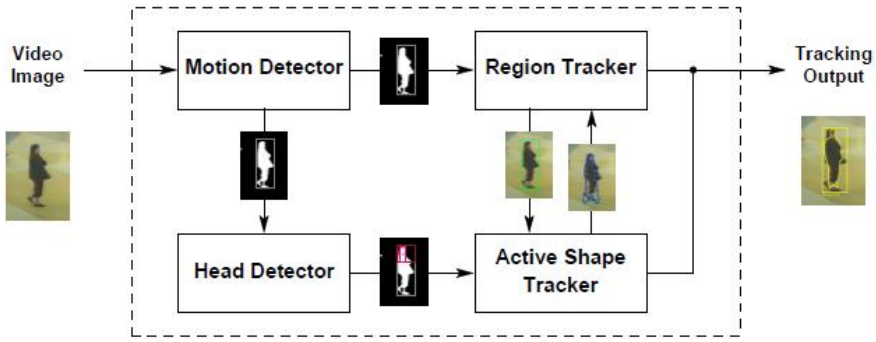


Figure 5.3: Overview of the four modules of the Reading People Tracker

those aspects of the tracking algorithms which are special to this system.

Figure 5.3 shows an overview of how the four modules comprise the People Tracker. Here is a brief overview of the functionalities of the individual models.

Motion Detector: This module models the background as an image with no people in it. The Background Image is subtracted pixelwise from the current video image and thresholded to yield the binary Motion Image. Regions with detected moving blobs are then extracted and written out as the output from this module.

Region Tracker: The Regions output by the Motion Detector are tracked over time by the Region Tracker. This includes region splitting and merging using predictions from the previous frame.

Head Detector: The Head Detector examines the areas of the binary Motion Image which correspond to moving regions tracked by the Region Tracker. The topmost points of the blobs in these region images that match certain criteria for size are output as possible positions of heads in these Regions.

Active Shape Tracker: This module uses an active shape model of 2D pedestrian outlines in the image to detect and track people. The initialisation of contour shapes is done from the output by the Region Tracker and the Head Detector.

The main goal of using more than one tracking module is to make up for deficiencies in the individual modules, thus achieving a better overall tracking performance than a single module could provide. Of course, when combining the information from different modules, it is important to be aware of the main sources of error for those modules. If two modules are subject to the same type of error, then there is little benefit in combining

the outputs. The new People Tracker has been designed keeping this aspect in mind, and using the redundancy introduced by the multiplicity of modules in an optimal manner. These are the main features of the system:

- interaction between modules to avoid non- or mis-detection
- independent prediction in the two tracking modules, for greater robustness
- multiple hypotheses during tracking to recover from lost or mixed-up tracks
- all modules have camera calibration data available for their use
- through the use of software engineering principles for the software design, the system is scalable and extensible (new modules...) as well as highly maintainable and portable

Each of the modules has access to the output from the modules run previously, and to the long-term tracking history which includes past and present tracks, together with the full tracking status (visibility, type of object, whether it is static etc). For each tracked object, all measurements, tracking hypotheses and predictions generated by different modules are stored in one place.

So, for all the reasons described above, the *Reading People Tracker* has been chosen as Image Processing Library of our prototype implementation. It is very easy to be used and it takes the frame sequence of the video as inputs and returns an XML file describing the low level annotations catched in each frame, according to a standard schema defined in a *XML Schema*. We have only made some few updates to the RPT's source code, in order to be able to get more easily the type of each object detected in a frame (person, package, car). For instance, figure 5.5 shows the low level annotations associated to the frame number 18 (figure 5.4) of a video belonging to the *ITEA - CANDELA* dataset (http://www.multitel.be/va/candela/abandon.html), which has been used to make some prelimary experiments.

As we can see in figure 5.5, the RPT correctly identifies two objects into the frame shown in figure 5.4: the former, identified by $ID = 5$, is a person, while the latter, identified by $ID = 100$ is a package.

However, we have manually filtered some errors found into low level annotations, in order to make a more reliable, correct and, first of all, unconditional experimentation of

Figure 5.4: A video frame from ITEA-CANDELA dataset

```
<frame xmlns="http://www.cvg.cs.reading.ac.uk/ADVISOR/people"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.cvg.cs.reading.ac.uk/ADVISOR
                                http://www.cvg.cs.reading.ac.uk/~nts/ADVISOR/people_tracker-multi.xsd"
       id ="18" pc_name = "fabio-desktop" num_cameras ="1">
  <camera id ="2002" time ="3600">
    <mobile id="5" start_time="3400">
      <track id="5" type="5">
        <info2d xmin="314" xmax="357" ymin="126" ymax="277" xcog="335" ycog="195" />
        <info3d x="0" y="0" z="0" width="0" height="0" />
        <occlusion left="0" right="0" bottom="0" top="0" />
      </track>
      <track id="100" type="6">
        <info2d xmin="314" xmax="357" ymin="126" ymax="277" xcog="333" ycog="197" />
        <info3d x="0" y="0" z="0" width="0" height="0" />
        <occlusion left="0" right="0" bottom="0" top="0" />
      </track>
    </mobile>
  </camera>
</frame>
```

Figure 5.5: The related low level annotations

our *Unexplained Activity Detector* on real-world datasets, that is the main goal of this thesis.

5.1.1.2 The Video Labeler

As we said before, the *Video Labeler* fills the semantic gap between the low level annotations captured for each frame and the high level annotations. So, through the Video Labeler, the *observation IDs* (as defined in the chapter 3) with the related *action symbols* and *timestamps* are detected; thus, the output of the Video Labeler is the *observation sequence* related to the considered video source.

The Video Labeler has been implemented in *Java* programming language: it uses the *DOM libraries* to parse the XML file containing the output of the *Image Processing Library*. The Video Library defines the rules that have to be checked to verify the presence of each interested *high level atomic event* in the video. So, a Java method for each action symbol we want to detect, containing the related rules, has been defined.

There are listed below some examples of rules defined to detect some *atomic events* (action symbols) in a video belonging to the *ITEA-CANDELA* dataset.

Action Symbol *A*: A person *P* goes into the central zone with the package

- There are at least two objects in the current frame
- At least one of the objects is a person
- At least one of the objects is a package
- The person identified appears on the scene for the first time
- The distance between the person's barycenter and the package one is smaller than an apposite distance threshold

Action Symbol *B*: A preson *P* leaves the package

- There are at least two objects in the current frame
- At least one of the objects is a person
- At least one of the objects is a package
- The person was previously holding a package
- The distance between the person's barycenter and the package one is smaller than an apposite distance threshold

Action Symbol *C*: A preson *P* goes into the central zone

- There are at least one object in the current frame
- At least one of the objects is a person
- The person identified appears on the scene for the first time

- If there are also some packages on the scene, their distances are greater than an apposite distance threshold

Action Symbol *D*: A preson *P* takes the package

- There are at least two objects in the current frame
- At least one of the objects is a person
- At least one of the objects is a package
- The distance between the person's barycenter and the package one is smaller than an apposite distance threshold
- The person was not previously holding a package

Action Symbol *E*: A person *P1* gives the package to another person *P2*

- There are at least three objects in the current frame
- At least two of the objects are persons
- At least one of the objects is a package
- P1 was previously holding a package
- In the current frame, both the distances of P1 and P2's barycenters from the package are smaller than an apposite distance threshold
- In the next frames, P1's distance from the package is greater than the threshold, while P2's one is smaller (it means that P2 has got the package and P1 is not holding it anymore)

Action Symbol *F*: A person *P* goes out of the central zone with the package

- This symbol is detected when a person holding a package does not appear anymore on the scene for a specified TTL

Thus, the output of the *Video Labeler* is the list of the *action symbols* detected in the video with the related *timestamps*; it is encoded in *comma-separated value* format.

5.1.1.3 The Activity Detection Engine

An *Activity Detection Engine* is able to find activity occurrences matching the well-known models: thus, such a module takes as inputs the *observation sequence* previously caught by the *Video Labeler* and the *stochastic activity models*. So, taking into account that the activity models must follow the schema defined into chapter 3, a specific software able to detect instances matching such models in time-stamped observation data has been used. Such software, called *tMAGIC*, is the implementation of a theoretical model presented in [54].

As a matter of fact, the [54] approach addresses the problem of efficiently detecting occurrences of high-level activities from such interleaved data streams. In this approach, there has been proposed a temporal probabilistic graph so that the elapsed time between observations also plays a role in defining whether a sequence of observations constitutes an activity. First, a data structure called *temporal multiactivity graph* to store multiple activities that need to be concurrently monitored has been proposed. Then, there has been defined an index called *Temporal Multi-Activity Graph Index Creation* (tMAGIC) that, based on this data structure, examines and links observations as they occur. There are also defined some algorithms for insertion and bulk insertion into the tMAGIC index and show that this can be efficiently accomplished. In this approach, the algorithms are basically defined to solve two problems: the *evidence problem* that tries to find all occurrences of an activity (with probability over a threshold) within a given sequence of observations, and the *identification problem* that tries to find the activity that best matches a sequence of observations. There are introduced some complexity reducing restrictions and pruning strategies to make the problem, which is intrinsically exponential, linear to the number of observations. It is demonstrated that tMAGIC has time and space complexity linear to the size of the input, and can efficiently retrieve instances of the monitored activities.

Thus, the output of the *Activity Detection Engine* are the well-known activity occurrences matching the defined models and their probabilities: both have been efficiently computed through the *tMAGIC* software.

5.1.1.4 The UAP Engine

As we have described in the previous chapters, the *UAP Engine* takes as input the activity occurrences previously found by the *Activity Detection Engine* with the associated probabilities and the observation sequence and finally discovers the *Unexplained Video Activities*. Such a module has been developed in Java programming language and provides the implementations of algorithms Top-k TUA, Top-k PUA, Top-k TUC. As we can also see in figure 5.1, the Java module uses some apposite APIs to interact with a *linear* and a *non-linear* program solver. More in details, the *QSopt* Library has been used for solving *linear programs* and the *Lingo* Library for *non-linear* ones.

Thus, the subsections 5.1.2 and 5.1.3 show the experimental evaluations we have made in the video surveillance domain. In particular, we evaluated our framework on two video datasets: (i) a video we shot by monitoring a university parking lot, and (ii) a benchmark dataset about video surveillance in a train station [55]. The frame observations have been generated in a semi-automatic way using both our prototype implementation we have just described and few human interventions. We have noted that identifying frame observations via the development of image processing algorithms is an extremely challenging task—the goal of our work is to present a domain-independent way of identifying unexplained activities that builds upon domain-specific ways of recognizing actions in observation sequences. In contrast to the difficulty of detecting actions in video, in cyber-security, it is easy to identify actions in an observation sequence as they can merely be logged.

5.1.2 Parking lot surveillance video

The set $\mathcal{A}$ includes "known" *normal activities* such as parking a car, people passing, a person getting in a car and leaving the parking lot, and *abnormal activities* such as a person taking a package out of the car and leaving it in the parking lot before driving away, or a person taking an unattended package in the parking lot. For instance, a model of a well-known *normal activity* is shown in figure 5.6.

Examples of detected unexplained activities are two cars stopping next to each other in the middle of the parking lot with the drivers exchanging something before leaving the parking lot, or a person strolling around a car for a while before leaving the parking lot.

We compared Algorithms Top-k TUA and Top-k PUA against "naïve" algorithms

Figure 5.6: Example of a well-known activity model on the parking lot dataset

which are the same as Top-k TUA and Top-k PUA but do not exploit the optimizations provided by the theorems in Chapter 3.2.1.

Figures 5.7 and 5.8 show that Top-k TUA and Top-k PUA significantly outperform the naïve algorithms which are not able to scale beyond videos of length 15 and 10 minutes for totally and partially unexplained activities, respectively (with longer videos, the naïve algorithms did not terminate in 3 hours). Figures 5.9(a) and 5.10(a) zoom in on the running times for Algorithms Top-k TUA and Top-k PUA, respectively. The runtimes in Figure 5.7 when $k = 5$ and $k = All$ are almost the same (the two curves are indistinguishable) because, up to 15 minutes, there were at most 5 totally unexplained activities in the video. A similar argument applies to Figure 5.8.

We also evaluated how the different parameters that define a UAP instance affect the running time by varying the values of each parameter while keeping the others fixed to a default value.

Runtime of Top-k TUA. Table 5.1 reports the values we considered for each parameter along with the corresponding default value.

For example, Table 5.1 says that we measured the running times to find the top-1, top-2, top-5, and all totally unexplained activities (as the video length increases) while

Parameter	Values	Default value
k	1, 2, 5, All (Top-k TUA)	All
	1, 5, 10, All (Top-k PUA)	All
τ	0.4, 0.6, 0.8	0.6
L	160, 200, 240, 280	200
# worlds	7 E+04, 4 E+05, 2 E+07	2 E+07

Table 5.1: Parameter values (parking lot dataset).

keeping $\tau = 0.6$, $L = 200$, $\#worlds = 2E + 07$.

It is important to note that another relevant parameter has also to be considered, that is the *frame rate*. It represents the frequency (rate) at which an imaging device produces unique consecutive images called frames. We can easily deduce that, the higher is the frame rate, the more complicated is the video, as we need a greater number of frames per second to represent its informative content. For what concerns the parking lot video, the frame rate has been set to *4* (4 frames per second). We want also to emphasize that the sequence length *L* is measured in frames: so, it is very useful to know the used frame rate, in order to get precise timing information.

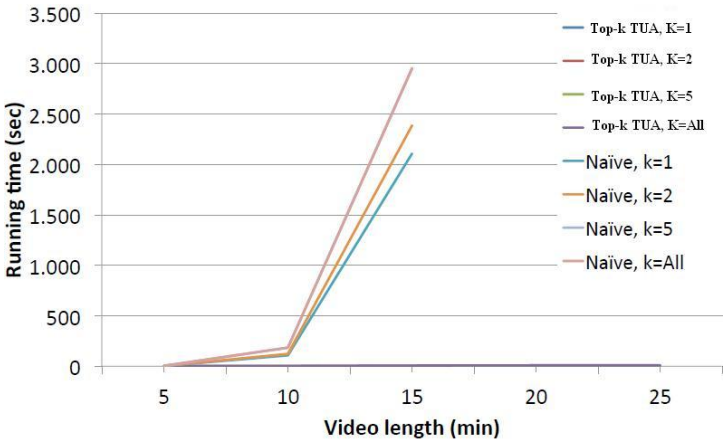


Figure 5.7: Algorithm Top-k TUA vs. Naïve.

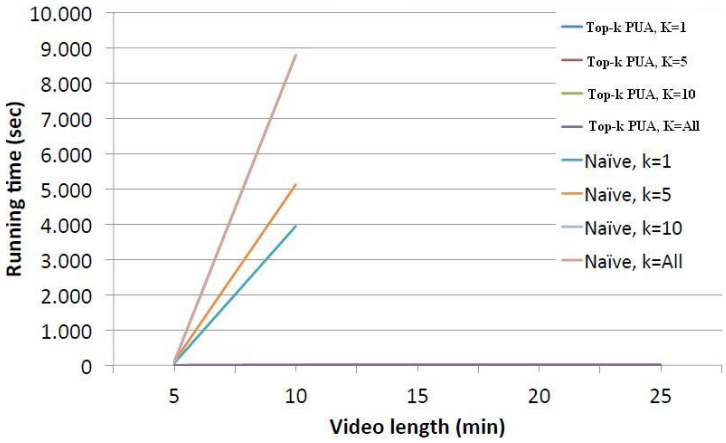


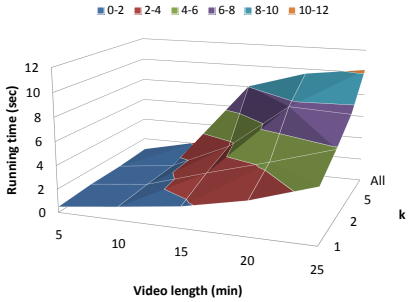
Figure 5.8: Algorithm Top-k PUA vs. Naïve.

Varying k. Figure 5.9(a) shows that lower values of k give lower runtimes. As discussed in the preceding chapter, Algorithm Top-k TUA can infer that some sequences are not going to be top-k TUAs and quickly prune: this is effective with lower values of k because the probability threshold to enter the current Top-k TUAs (i.e., $lowest$ in Algorithm Top-k TUA) is higher, thus fewer candidates are added to the current Top-k TUAs, making the pruning of Algorithm Top-k TUA more effective.

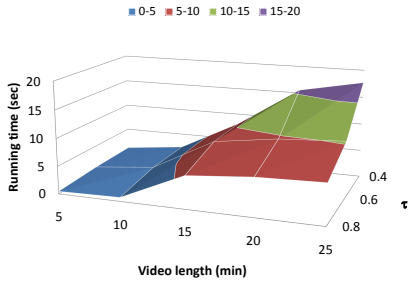
Varying τ. Figure 5.9(b) shows that the runtime decreases as the probability threshold grows. Intuitively, this is because higher probability thresholds enable Algorithm Top-k TUA to prune more.

Varying L. Figure 5.9(c) shows that higher values of L yield lower running times, though there is not a big difference between $L = 200$ and $L = 240$.

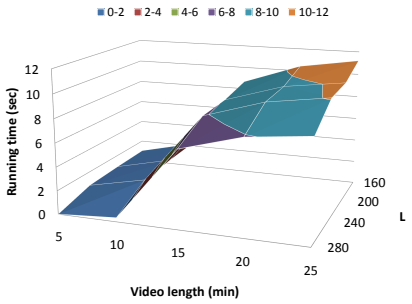
Varying Number of Possible Worlds. Finally, Figure 5.9(d) shows that more possible



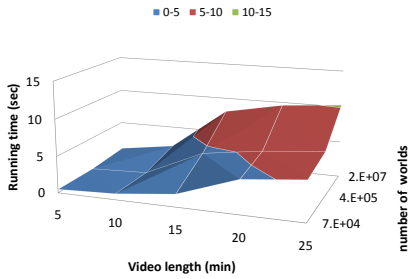
((a)) Varying k ($\tau = 0.6, L = 200$)



((b)) Varying τ ($k = All, L = 200$)



((c)) Varying L ($\tau = 0.6, k = All$)



((d)) Varying number of worlds ($\tau = 0.6, k = All, L = 200$)

Figure 5.9: Running time of Algorithm Top-k TUA on the parking lot dataset.

worlds leads to higher running times. However, note that big differences in the number of possible worlds yield small differences in running times, hence Algorithm Top-k TUA is able to scale well (this is due to the application of Theorem 2 to compute $\mathcal{P}_T(S)$).

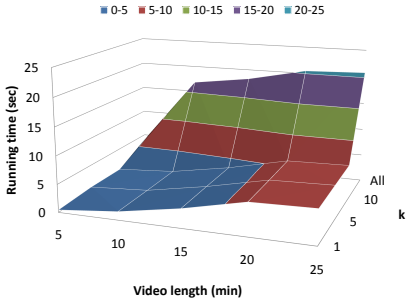
Runtime of Top-k PUA. The parameter values we used are reported in Table 5.1.

Varying k. The runtimes for $k = 1, 5, 10$ differ slightly from each other and are much lower than when all PUAs had to be found (Figure 5.10(a)).

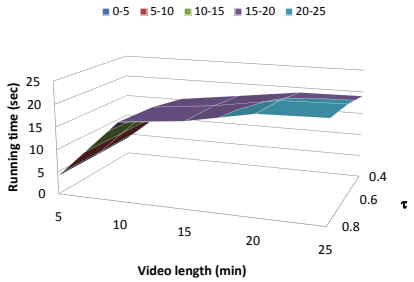
Varying τ. Figure 5.10(b) shows that the runtimes do not change much for different values of τ.

Varying L. Figure 5.10(c) shows that higher values of L lead to lower runtimes.

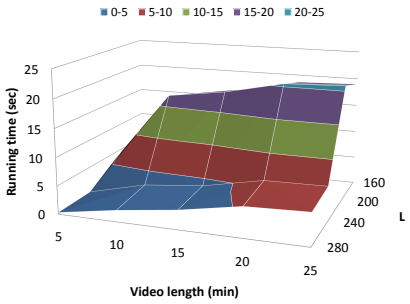
Varying Number of Possible Worlds. Figure 5.10(d) shows that higher numbers of possible worlds lead to higher runtimes. As with TUAs, the runtime of Top-k PUA increases reasonably despite the steep growth of possible worlds. Runtimes of Top-k



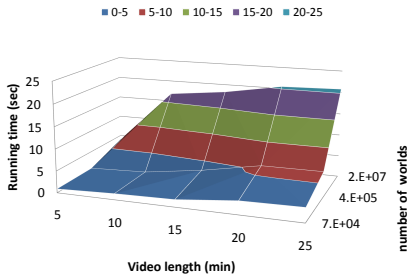
((a)) Varying k ($\tau = 0.6, L = 200$)



((b)) Varying τ ($k = All, L = 200$)



((c)) Varying L ($\tau = 0.6, k = All$)



((d)) Varying number of worlds ($\tau = 0.6, k = All, L = 200$)

Figure 5.10: Running time of Algorithm Top-k PUA on the parking lot dataset.

PUA are higher than for Top-k TUA because computing $\mathcal{P}_P(S)$ requires solving a non-linear program whereas $\mathcal{P}_T(S)$ requires solving linear programs.

Precision/Recall. In order to assess accuracy, we compared the output of our algorithms against ground truth provided by 8 human annotators who were taught the meaning of graphical representations of activities in $\mathcal{A}$ (e.g., Figure 3.1). They were asked to identify the totally and partially unexplained activities w.r.t. $\mathcal{A}$. We ran Top-k TUA and Top-k PUA with values of τ ranging from 0.4 to 0.8, looking for *all* totally and partially unexplained activities (L was set to 200). We use $\{S_i^a\}_{i\in[1,m]}$ to denote the unexplained sequences returned by our algorithms and $\{S_j^h\}_{j\in[1,n]}$ to denote the sequences flagged as unexplained by human annotators. Precision and recall were computed as follows:

$$P = \frac{|\{S_i^a | \exists S_j^h \; s.t. \; S_i^a \approx S_j^h\}|}{m} \tag{5.1}$$

and

$$R = \frac{|\{S_j^h | \exists S_i^a \; s.t. \; S_i^a \approx S_j^h\}|}{n} \tag{5.2}$$

where $S_i^a \approx_p S_j^h$ means that S_i^a and S_j^h overlap by a percentage no smaller than 75%.

Precision and recall when $\tau = 0.4, 0.6, 0.8$ are shown in Tables 5.1(a) and 5.1(b) and Figure 5.11: we can easily notice that, the higher is the probability threshold value, the higher is the precision, the lower is the recall and vice versa. That is exactly what we reasonably expected. In summary, we can say that our framework achieved a good accuracy.

((a)) Top-k TUA

τ	Precision	Recall
0.4	62.5	89.17
0.6	66.67	82.5
0.8	72.22	71.67

((b)) Top-k PUA

τ	Precision	Recall
0.4	59.65	77.38
0.6	64.91	74.6
0.8	70.18	71.83

Table 5.2: Precision and recall (parking lot dataset).

5.1.3 Train station surveillance video

We also tested our algorithms with a train station video surveillance dataset [55]. The set $\mathcal{A}$ of known activities includes normal activities such as people passing, people chatting,

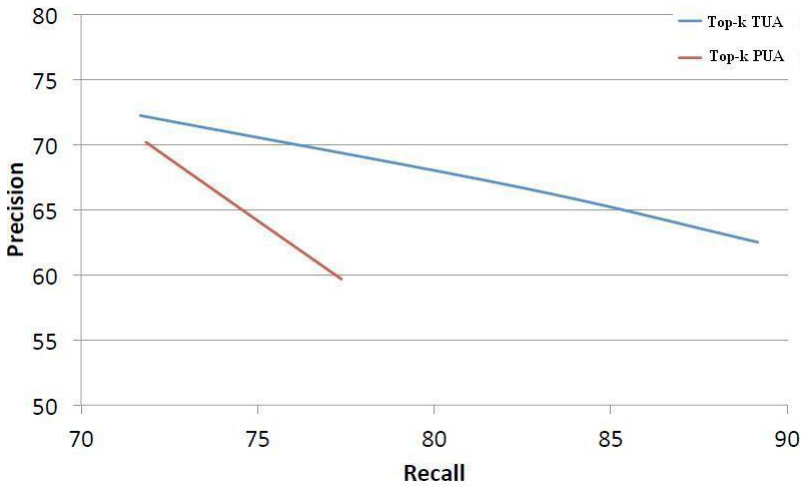


Figure 5.11: Precision/Recall on the parking lot dataset

people seating, and abnormal activities like a person leaving a package unattended, or a person taking an unattended package. For instance, a model of a well-known activity is shown in figure 5.12.

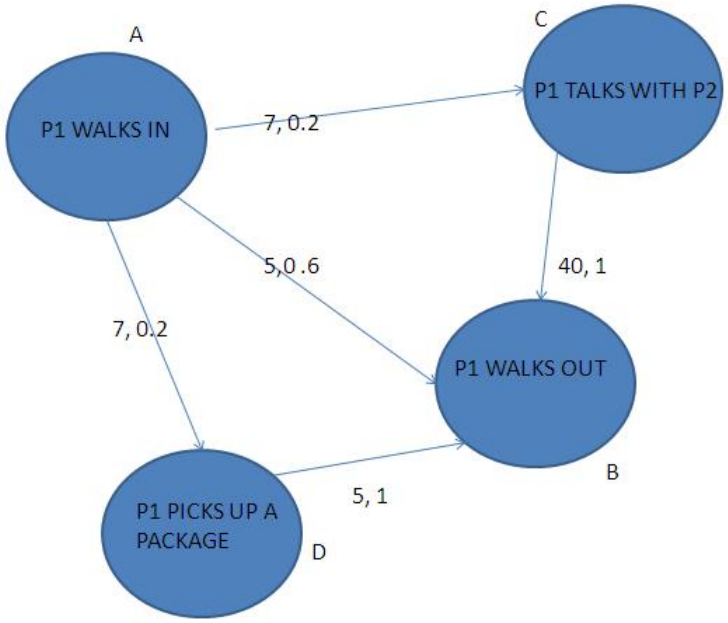
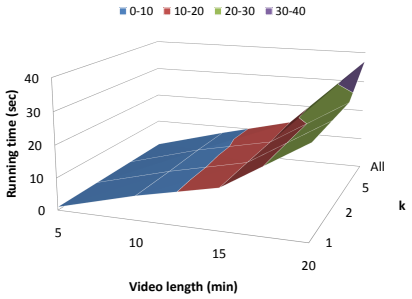


Figure 5.12: Example of a well-known activity model on the train station dataset

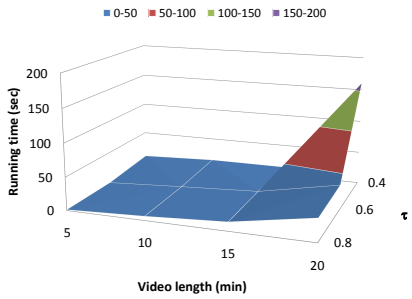
Examples of found unexplained activities are a person typing on a keypad next to a door with the door closing afterwards, or a person leaving a suitcase on the ground, standing next to it for a while looking at his watch different times, taking the suitcase, and leaving the area.

Runtime of Top-k TUA. This data set is far more complex (w.r.t. number of possible worlds) than the parking lot data set (thus, the frame rate has been set to *25*). For this reason, the "naïve" algorithms did not terminate in a reasonable amount of time, even with a video of 5 minutes. Thus, we do not show the runtimes of the naïve algorithms. As in the case of the parking lot data set, we varied the k, τ, L, $\#\ worlds$ parameters, as shown in Table 5.3.

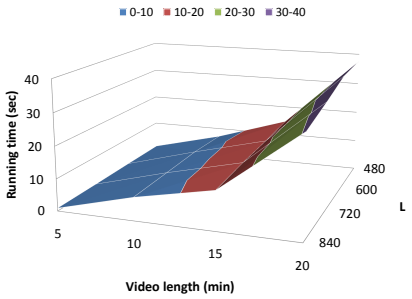
Varying k. Figure 5.13(a) shows that Top-k TUA's runtime varies little with k when the video is up to 15 minutes long. After that, the runtime for $k = 1, 2, 5$ are comparable,



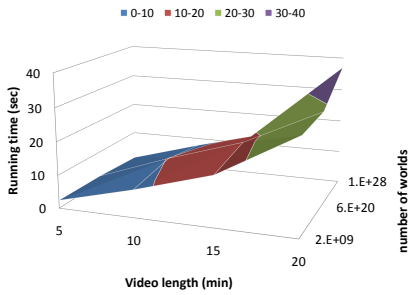
((a)) Varying k ($\tau = 0.6, L = 600$)



((b)) Varying τ ($k = All, L = 600$)



((c)) Varying L ($\tau = 0.6, k = All$)



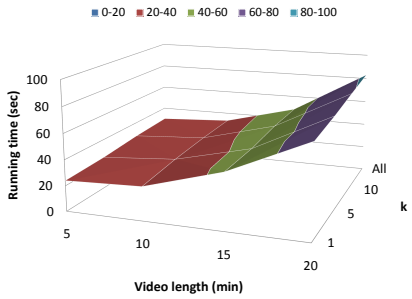
((d)) Varying number of worlds ($\tau = 0.6, k = All, L = 600$)

Figure 5.13: Running time of Algorithm Top-k TUA on the train station dataset.

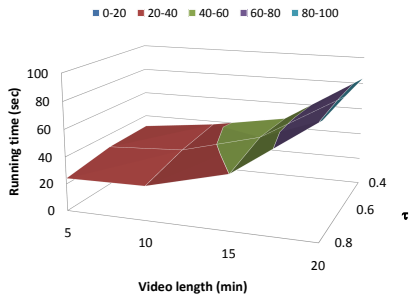
but the runtime for $k = All$ starts to diverge from them.

Varying τ. Figure 5.13(b) shows that the runtime when $\tau = 0.4$ is much higher than when $\tau = 0.6$ and $\tau = 0.8$ (the latter two cases do not show substantial differences in running time).

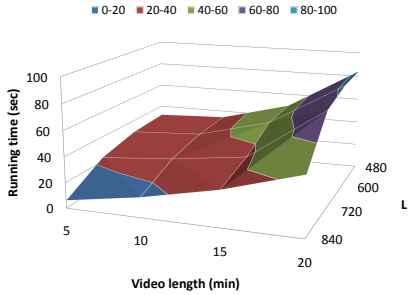
Varying L. Figure 5.13(c) shows that higher values of L yield lower runtimes. Though



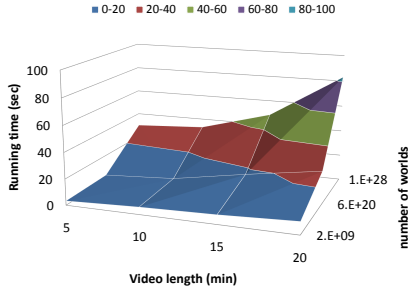
((a)) Varying k ($\tau = 0.6, L = 600$)



((b)) Varying τ ($k = All, L = 600$)



((c)) Varying L ($\tau = 0.6, k = All$)



((d)) Varying number of worlds ($\tau = 0.6, k = All, L = 600$)

Figure 5.14: Running time of Algorithm Top-k PUA on the train station dataset.

Parameter	Values	Default value
k	1, 2, 5, All (Top-k TUA)	All
	1, 5, 10, All (Top-k PUA)	All
τ	0.4, 0.6, 0.8	0.6
L	480, 600, 720, 840	600
# worlds	2 E+09, 6 E+20, 1 E+28	1 E+28

Table 5.3: Parameter values (train station dataset).

the difference is small for videos under 15 minutes, it increases for 20 minute videos.

Varying Number of Possible Worlds. Figure 5.13(d) shows that runtimes for different numbers of possible worlds are initially close (up to 15 minutes); then, the runtime for 1 E+28 possible worlds gets higher. There is only a moderate increase in runtime corresponding to a huge increase of the number of possible worlds—hence, Top-k TUA is able to scale well when the video gets substantially more complex.

Runtime of Top-k PUA. We varied the k, τ, L, $\#\ worlds$ parameters as reported in Table 5.3.

Varying k. Figure 5.14(a) shows that the runtime decreases as k decreases.

Varying τ. Figure 5.14(b) shows that the runtimes for $\tau = 0.4$ and $\tau = 0.6$ are similar and higher than the one for $\tau = 0.8$.

Varying L. Figure 5.14(c) shows that lower values of L give higher running times. The runtimes are similar for $L = 480$ and $L = 600$ (the number of PUAs found in the video are similar in both cases). Execution times are lower for $L = 720$ and much lower for $L = 800$ (in this case, the number of PUAs found in the video is approximately half the number of PUAs found with $L = 480$ and $L = 600$).

Varying Number of Possible Worlds. Figure 5.14(d) shows that though the runtime grows with the number of possible worlds, Top-k PUA responds well to the steep growth of the number of possible worlds.

Precision/Recall. We evaluated the accuracy of Top-k TUA and Top-k PUA in the same way as for the parking lot data set. Precision and recall are reported in Tables 5.3(a), 5.3(b) and Figure 5.15. We can easily notice also in this case study that, the higher is the probability threshold value, the higher is the precision, the lower is the recall and vice versa. In summary, the obtained results show that we achieved high accuracy.

((a)) Top-k TUA

τ	Precision	Recall
0.4	56.48	80.35
0.6	78.79	76.25
0.8	81.82	73.99

((b)) Top-k PUA

τ	Precision	Recall
0.4	72.62	77.12
0.6	75	73.59
0.8	76.19	71.5

Table 5.4: Precision and recall (train station dataset).

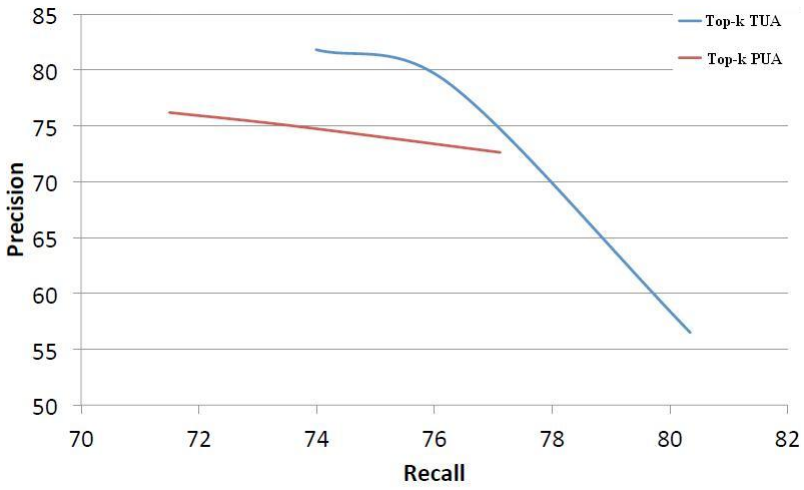


Figure 5.15: Precision/Recall on the train station dataset

5.2 Cyber security

As we highlighted before, in contrast to the difficulty of detecting actions in video, in cyber-security it is easy to identify actions in an observation sequence as they can merely be logged.

Anyway, in order to execute an extended and complete experimentation on a real-world dataset, we have designed and developed a specific prototype implementation for cyber security domain. Such a prototype is described in detail in subsection 5.2.1.

5.2.1 The developed prototype

As we can see in figure 5.16, our prototype consists of:

- A network Sniffer
- A network Intrusion Detection System
- An Alert Aggregation module
- The UAP Engine

In particular, the *Sniffer* captures network traffic and generates the sequence of packets, the *Intrusion Detection System* analyzes such traffic and generates the sequence of

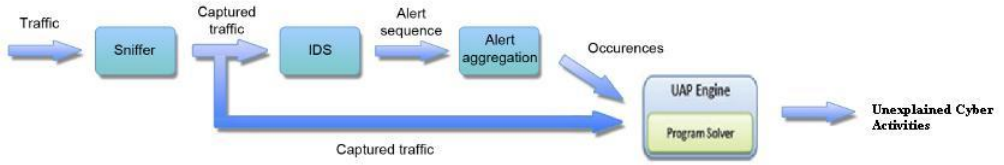


Figure 5.16: The prototype architecture for cyber security context

alerts. Then, as the number of alerts returned by the IDS may be relatively high, the *Alert Aggregation module*, that takes as input the identified alerts, can optionally aggregate multiple alerts triggered by the same event into a macro-alert, based on a set of ad hoc aggregation rules. For instance, we defined rules to aggregate alerts such that protocol, source address, and destination address of suspicious traffic are the same, and the alerts are within a given temporal window. In other words, the events triggering such alerts will be treated as a single event, thus reducing the amount of data to be processed. Finally, our *UAP Engine* takes as inputs the occurrences detected during the previous step and the whole captured traffic as well and discovers the *unexplained cyber activities*.

All these modules are described in the next subsections (5.2.1.1, 5.2.1.2, 5.2.1.3, 5.2.1.4).

5.2.1.1 Sniffer

The Sniffer chosen for our prototype has been *Wireshark* (*http://www.wireshark.org/*). As a matter of fact, it is a *free and open-source packet analyzer*. It is used for *network* troubleshooting, analysis, software and *communications protocol* development, and education. Originally named *Ethereal*, in May 2006 the project was renamed Wireshark due to trademark issues.

Wireshark is *cross-platform*, using the *GTK+ widget toolkit* to implement its user interface, and using *pcap* to capture packets; it runs on various *Unix-like* operating systems including *Linux*, *OS X*, *BSD*, and *Solaris*, and on *Microsoft Windows*. There is also a terminal-based (non-GUI) version called *TShark*. Wireshark, and the other programs distributed with it such as TShark, are *free software*, released under the terms of the *GNU General Public License*.

Wireshark is very similar to *tcpdump*, but has a graphical front-end, plus some integrated sorting and filtering options. It allows the user to put *network interface controllers*

that support promiscuous mode into that mode, in order to see all traffic visible on that interface, not just traffic addressed to one of the interface's configured addresses and broadcast/multicast traffic. However, when capturing with a packet analyzer in promiscuous mode on a port on a *network switch*, not all of the traffic traveling through the switch will necessarily be sent to the port on which the capture is being done, so capturing in promiscuous mode will not necessarily be sufficient to see all traffic on the network. *Port mirroring* or various *network taps* extend capture to any point on net; simple passive taps are extremely resistant to *malware tampering*.

On Linux, BSD, and OS X, with *libpcap* 1.0.0 or later, Wireshark 1.4 and later can also put *wireless network interface controllers* into *monitor mode*.

In the late 1990s, Gerald Combs, a computer science graduate of the *University of Missouri, Kansas City*, was working for a small *Internet service provider*. The commercial protocol analysis products at the time were priced around $1500 and did not run on the company's primary platforms (Solaris and Linux), so Gerald began writing Ethereal and released the first version around 1998. The Ethereal *trademark* is owned by Network Integration Services. In May 2006, Combs accepted a job with CACE Technologies. Combs still held copyright on most of Ethereal's source code (and the rest was redistributable under the GNU GPL), so he used the contents of the Ethereal *Subversion* repository as the basis for the Wireshark repository. However, he did not own the Ethereal trademark, so he changed the name to Wireshark. In 2010 *Riverbed Technology* purchased CACE and took over as the primary sponsor of Wireshark. Ethereal development has ceased, and an Ethereal security advisory recommended switching to Wireshark. Wireshark has won several industry awards over the years, including *eWeek*, *InfoWorld* and *PC Magazine*. It is also the top-rated packet sniffer in the Insecure.Org network security tools survey and was the *SourceForge* Project of the Month in August 2010. Combs continues to maintain the overall code of Wireshark and issue releases of new versions of the software. The product website lists over 600 additional contributing authors.

Wireshark is software that "understands" the structure of different networking protocols. Thus, it is able to display the encapsulation and the fields along with their meanings of different packets specified by different networking protocols. Wireshark uses *pcap* to capture packets, so it can only capture the packets on the types of networks that pcap supports.

The main features of Wireshark are listed below:

- Data can be captured ”from the wire” from a live network connection or read from a file that recorded already-captured packets.
- Live data can be read from a number of types of network, including *Ethernet*, *IEEE 802.11*, *PPP*, and *loopback*.
- Captured network data can be browsed via a *GUI*, or via the terminal (*command line*) version of the utility, TShark.
- Captured files can be programmatically edited or converted via command-line switches to the ”editcap” program.
- Data display can be refined using a display filter.
- *Plug-ins* can be created for dissecting new protocols.
- *VoIP* calls in the captured traffic can be detected. If encoded in a compatible encoding, the media flow can even be played.
- Raw *USB* traffic can be captured.

Wireshark's native network trace file format is the libpcap format supported by *libpcap and WinPcap*, so it can exchange files of captured network traces with other applications using the same format, including *tcpdump* and *CA NetMaster*. It can also read captures from other network analyzers, such as *snoop*, *Network General*'s Sniffer, and *Microsoft Network Monitor*.

Capturing raw network traffic from an interface requires elevated privileges on some platforms. For this reason, older versions of Ethereal/Wireshark and tethereal/TShark often ran with *superuser* privileges. Taking into account the huge number of protocol dissectors that are called when traffic is captured, this can pose a serious security risk given the possibility of a bug in a dissector. Due to the rather large number of vulnerabilities in the past (of which many have allowed remote code execution) and developers' doubts for better future development, *OpenBSD* removed Ethereal from its ports tree prior to OpenBSD 3.6.

Elevated privileges are not needed for all of the operations. For example, an alternative is to run *tcpdump*, or the dumpcap utility that comes with Wireshark, with superuser

privileges to capture packets into a file, and later analyze the packets by running Wireshark with restricted privileges. To make near real time analysis, each captured file may be merged by mergecap into growing file processed by Wireshark. On wireless networks, it is possible to use the *Aircrack* wireless security tools to capture *IEEE 802.11* frames and read the resulting dump files with Wireshark.

As of Wireshark 0.99.7, Wireshark and TShark run dumpcap to do traffic capture. On platforms where special privileges are needed to capture traffic, only dumpcap needs to be set up to run with those special privileges: neither Wireshark nor TShark need to run with special privileges, and neither of them should be run with special privileges.

5.2.1.2 Intrusion Detection System

We have chosen *Snort* (*http://www.snort.org/*) as Intrusion Detection System in our prototype implementation.

Snort is an open source network intrusion prevention and detection system (IDS/IPS) developed by Sourcefire. Combining the benefits of signature, protocol, and anomaly-based inspection, Snort is the most widely deployed IDS/IPS technology worldwide. It contains many configurable internal components that can vastly influence false positives and negatives as well as general packet logging performance. Understanding the function of these internal components will help customize Snort to the analyzed network and help to avoid some of the common Snort pitfalls. Snort can be divided into five major components that are each critical to intrusion detection. The first is the *packet capturing mechanism*. Snort relies on an external packet capturing library (libpcap) to sniff packets. After packets have been captured in a raw form, they are passed into the *packet decoder*. The decoder is the first step into Snorts own architecture. The packet decoder translates specific protocol elements into an internal data structure. After the initial preparatory packet capture and decode is completed, traffic is handled by the *preprocessors*. Any number of pluggable preprocessors either examine or manipulate packets before handing them to the next component: the *detection engine*. The detection engine performs simple tests on a single aspect of each packet to detect intrusions. The last component is the *output plugins*, which generate alerts to present suspicious activity to users. A simplified graphical representation of the dataflow is shown in Figure 5.17.

Now, let us briefly describe the five Snort components:

Packet Capturing Mechanism: to get packets into the preprocessors and then the main

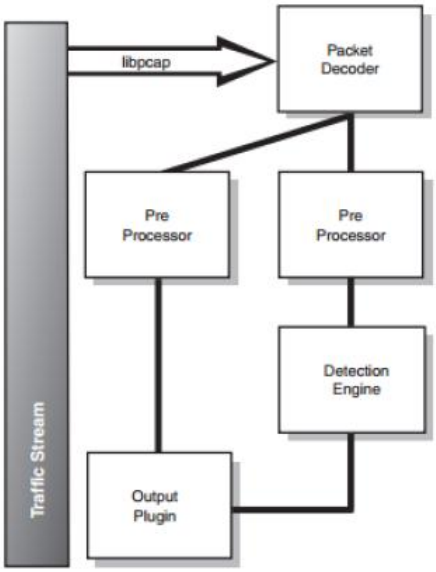


Figure 5.17: Snort component dataflow

detection engine, some prior labor must first occur. Snort has no native packet capture facility yet; it requires an external packet sniffing library: libpcap. Libpcap was chosen for packet capture for its platform independence. Using libpcap makes Snort a truly platform-independent application. The responsibility for grabbing packets directly from the network interface card belongs to libpcap. It makes the capture facility for raw packets provided by the underlying operating system available to other applications.

A raw packet is a packet that is left in its original, unmodified form as it had traveled across the network from client to server. A raw packet has all its protocol header information left intact and unaltered by the operating system. Network applications typically do not process raw packets; they depend on the OS to read protocol information and properly forward payload data to them. Snort is unusual in this sense in that it requires the opposite: it needs to have the packets in their raw state to function. Snort uses protocol header information that would have been stripped off by the operating system to detect some forms of attacks. Using libpcap is not the most efficient way to acquire raw packets. It can process only one packet at a time, making it a bottleneck for high-bandwidth (1Gbps) monitoring situations. In the future Snort will likely implement packet capture libraries specific to an OS, or even hardware.There are several methods other than us-

ing libpcap for grabbing packets from a network interface card. Berkeley Packet Filter (BPF), Data Link Provider Interface (DLPI), and the SOCK PACKET mechanism in the Linux kernel are other tools for grabbing raw packets.

Packet Decoder: as soon as packets have been gathered, Snort must decode the specific protocol elements for each packet. The packet decoder is actually a series of decoders that each decode specific protocol elements. It works up the Network stack, starting with lower level Data Link protocols, decoding each protocol as it moves up. A packet follows this data flow as it moves through the packet decoder 5.18. As packets move through the various protocol decoders, a data structure is filled up with decoded packet data. As soon as packet data is stored in a data structure it is ready to be analyzed by the preprocessors and the detection engine.

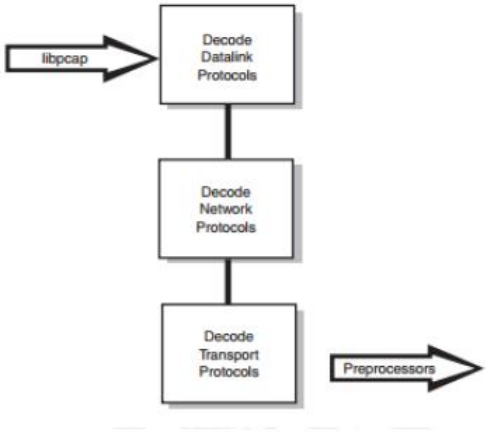


Figure 5.18: Decoder data flow

Preprocessor: Snorts preprocessors fall into two categories. They can be used to either examine packets for suspicious activity or modify packets so that the detection engine can properly interpret them. A number of attacks cannot be detected by signature matching via the detection engine, so examine preprocessors step up to the plate and detect suspicious activity. These types of preprocessors are indispensable in discovering non-signature-based attacks. The other preprocessors are responsible for normalizing traffic so that the detection engine can accurately match signatures. These preprocessors defeat attacks that attempt to evade Snorts detection engine by manipulating traffic patterns.

Additionally, Snort cycles packets through every preprocessor to discover attacks

that require more than one preprocessor to detect them. If Snort simply stopped checking for the suspicious attributes of a packet after it had set off an alert via a preprocessor, attackers could use this deficiency to hide traffic from Snort. Suppose a black hat intentionally encoded a malicious remote exploit attack in a manner that would set off a low priority alert from a preprocessor. If processing is assumed to be finished at this point and the packet is no longer cycled through the preprocessors, the remote exploit attack would register only an encoding alert. The remote exploit would go unnoticed by Snort, obscuring the true nature of the traffic. Preprocessor parameters are configured and tuned via the *snort.conf* file.

Detection Engine: is the primary Snort component. It has two major functions: *rules parsing* and *signature detection*. The detection engine builds attack signatures by parsing Snort rules. Snort rules are read line by line, and are loaded into an internal data structure. The rules are loaded only when the Snort service is started, meaning that to modify, add, or delete a rule, the user must refresh the Snort daemon. The detection engine runs traffic through the now loaded rule set in the order that it loads them into memory. It is possible to dictate which rules are run first by prioritizing. Rules are split into two functional sections:

- *Rule header*: contains information about the conditions for applying the signature.
- *Rule option*: for the same exploit begins and ends with a parenthetical. The rule option contains the actual signature, the priority level, and some documentation about the attack.

The detection engine processes rule headers and rule options differently. The detection engine builds a linked list decision tree. The nodes of the tree test each incoming packet for increasingly precise signature elements. A packet is tested to see whether it is TCP; if so, it is passed to the portion of the tree that has rules for TCP. The packet is then tested to see whether it matches a source address in a rule; if so, it passes down the corresponding rule chains. This process happens until the packet either matches an attack signature or tests clean and is dropped. The important thing to remember is that Snort commences testing a packet after it has found a signature to match to the packet. Even if the packet could possibly match another signature, the detection engine moves on to the next packet. This is why it is valuable to organize rules so that the most malicious signatures are loaded first.

Output plugins: Snort's output plugins are the means Snort has to get intrusion data to users. The purpose of the output plugins is to dump alerting data to another resource or file. Multiple outputting plugins can be activated to perform different functions. Loads of external applications - some even built exclusively for Snort - are designed to read Snorts output and manage intrusion data. Output plugins can be a major bottleneck for Snort. Snort can read and process packets quickly, but bogs down when trying to write to a slow database or over a network. Database output plugins are not used in high-bandwidth environments. It is recommended to configure Snort to spool to unified format and let Snort's unified log application, Barnyard, take over. Snort has 12 output plugins that push out data in different formats.

5.2.1.3 Alert Aggregation

The main purpose of using alert aggregation [41] is to reduce the redundancy of alerts by grouping duplicate alerts and merging them into a single one. Alerts are considered to be aggregated in terms of their attributes, such as *timestamp*, *source IP*, *target IP*, *port(s)*, *user name*, *process name*, *attack class* and *sensor ID*, as they are defined in IDMEF. Normally, the alerts raised by different sensors with the same attributes (e.g., timestamp, source IP, target IP, port(s), and attack class) can be merged together. Timestamps can be slightly different but should be close enough to fall into a predefined time window. A similar method used by Debar and Wespi in their Aggregation and Correlation Component (ACC) is called *duplicate relationship*. The duplicate relationships between alerts are defined in a duplicate definition file. Based on the definitions defined in such a file, the attributes of the new alerts are compared with those of the previous alerts for aggregation.

In addition to the aggregation, *alert compression* is another simple technique for dealing with duplicate alerts. These techniques use a Run Length Encoding (RLE) to represent a particular type of repeated alerts, that is, a recurring sequence of alerts is simply replaced by RLE with a single alert and a run count. In most cases, identifying redundant or duplicate alerts is not a difficult task, but for more complex cases, some predefined criteria are required. For example, if an attacker is scanning different ranges of the IP addresses in a network, this will trigger multiple alerts with different ranges of target IPs, but they should still be aggregated into an only alert.

In summary, our *Alert Aggregation module* takes as input the previously identified

alerts and can optionally aggregate multiple alerts triggered by the same event into a macro-alert, based on a set of ad hoc aggregation rules. The set of the used aggregation rules is described below.

Let us introduce the following variables:

m = number of bins

n = number of alerts

$$y_i = \begin{cases} 1, & \text{if the bin } i \text{ is used} \\ 0, & \text{otherwise} \end{cases} \quad \text{for i=1,...,m}$$

$$x_{ij} = \begin{cases} 1, & \text{if alert } j \text{ is assigned to bin } i \\ 0, & \text{otherwise} \end{cases} \quad \text{for i=1,...,m j=1,....,n}$$

$Source_j$ = Source IP address of alert j

$Destination_j$ = Destination IP address of alert j

$Protocol_j$ = Protocol of alert j

t_j = Arrival time of alert j

τ_g = global threshold

τ_l = local threshold

So, we can give a formal definition of the used aggregation rules.

Rule 1: two alerts *i* and *j* are aggregated if they satisfy the following condition:

$$\prec Source\_IP, Destination\_IP, Protocol \succ_i = \prec Source\_IP, Destination\_IP, Protocol \succ_j$$

So, the following is the mathematical model of Rule 1:

$$z = \sum_{i=1}^{m} y_i$$

$$s.t.$$

$$\begin{cases} t_j * x_{ij} \leq (t_1 + \tau_g) * x_{i1} \text{ i=1,...,m j=1,....,n} \\ t_j * x_{ij} \leq (t_{j-1} + \tau_l) * x_{ij-1} \text{ i=1,...,m j=1,....,n} \\ Source_j = Source_{j-1} \text{ j=1,....,n} \\ Destination_j = Destination_{j-1} \text{ j=1,....,n} \\ Protocol_j = Protocol_{j-1} \text{ j=1,....,n} \\ y_i = 0/1 \\ xij = 0/1 \end{cases}$$

Rule 2: two alerts *i* and *j* are aggregated if either:

$$\prec Source\_IP, Destination\_IP, Protocol \succ_i = \prec Source\_IP, Destination\_IP, Protocol \succ_j$$

or:

$$\prec Source\_IP, Destination\_IP, Protocol \succ_i = \prec Destination\_IP, Source\_IP, Protocol \succ_j$$

Thus, the output of the *Alert Aggregation* phase consists of a *macro-alert list* related to the analyzed network traffic and encoded in a comma separated value format.

5.2.1.4 The UAP Engine

As we have specified before, the *UAP Engine* takes as inputs the macro-alert list (corresponding to the occurrences of the well-known models, in the general model) detected by the *Alert Aggregation Module* and the whole captured traffic as well and discovers the unexplained cyber activities. Such a module is, of course, exactly the same as the video surveillance context.

Next subsection (5.2.2) describes our prelimary experiments conducted on a quite-simple *University* dataset.

5.2.2 The "University of Naples" dataset

We ran some preliminary evaluations with a cyber security dataset consisting of network traffic from *University of Naples* network.

We used the prototype implementation described before to get first packet sequences, then the alert sequences and finally the aggregated alerts. So, we ran our *UAP Engine*.

The list of the executed experiments is shown below.

Runtime. We varied the k, τ, L, $\#\ worlds$ parameters as shown in Table 5.5 and used *all* Snort rules as the set of activity models. As we can notice through the number of worlds listed in table 5.5, this dataset is definitely easier than the video surveillance ones, also because the *Alert Aggregation Module* significantly prunes the number of alerts which was at the beginning enormous. Moreover, it is clear that the *frame rate parameter* does not make sense in a cyber security context, so *L* is measured in milliseconds.

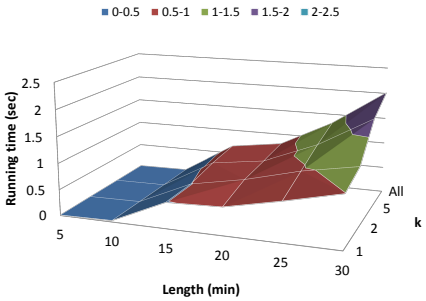
Parameter	Values	Default value
k	1, 2, 5, All (Top-k TUA)	All
	1, 3, 6, All (Top-k PUA)	All
τ	0.4, 0.6, 0.8	0.6
L	180000, 300000, 420000, 480000	300000
# worlds	2341, 65457, 897653	897653

Table 5.5: Parameter values (cyber security dataset).

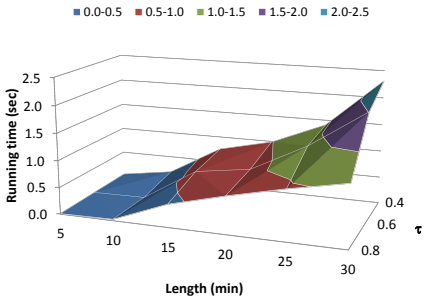
Running times for Algorithm Top-k TUA and Algorithm Top-k PUA are shown in Figures 5.19 and 5.20, respectively. They confirm the trend already seen with video data: runtime decreases as L and τ (resp. k and the number of possible worlds) increase (resp. decrease).

Accuracy. We measured accuracy as follows. Let $\mathcal{A}$ be the set of *all* Snort rules. First, we detected all occurrences of $\mathcal{A}$ in the data stream. We then ignored a certain subset $\mathcal{A}'$ of $\mathcal{A}$ and identified the unexplained sequences. Clearly, ignoring models in $\mathcal{A}'$ is equivalent to not having those models available. Thus, occurrences of ignored activties are expected to have a relatively high probability of being unexplained as there is no model for them. We measured the fraction of such occurrences that have been flagged as unexplained for different values of τ.

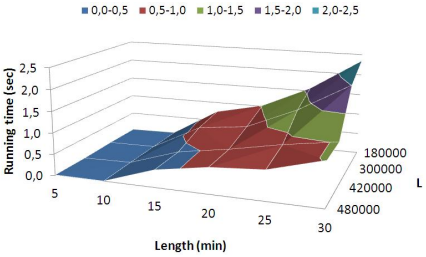
Specifically, we considered two settings: one where only *ICMP rules* in $\mathcal{A}$ were ignored, and another one where only *preprocessor rules* in $\mathcal{A}$ were ignored. ICMP rules are Snort rules designed to analyze ICMP packets (e.g., echo request a.k.a. ping) and alert on suspicious or malformed ICMP packets. For instance, a ping sweep is a passive reconnaissance attack that uses multiple echo requests to establish which IP addresses



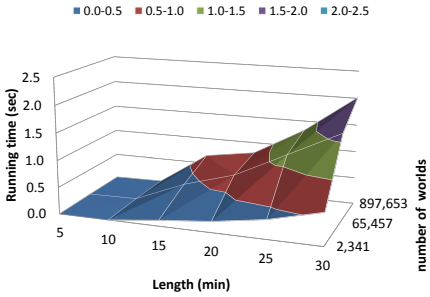
((a)) Varying k ($\tau = 0.6, L = 300000$)



((b)) Varying τ ($k = All, L = 300000$)



((c)) Varying L ($\tau = 0.6, k = All$)



((d)) Varying number of worlds ($\tau = 0.6, k = All, L = 300000$)

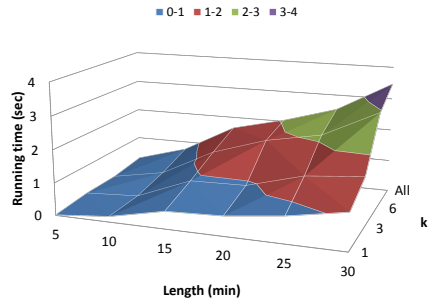
Figure 5.19: Running time of Algorithm Top-k TUA on the cyber security dataset.

map to live hosts. Some packets and applications have to be decoded into plain text for Snort rules to trigger. Prepocessor rules are designed to handle such situations. For instance, the *arpspoof* preprocessor is fed a list of IP:MAC addresses. When it detects a layer-2 attack, it triggers an alarm for a layer-2 event, such as multiple MAC addresses from a single IP. The results are reported in Tables 5.6 and 5.7 for Top-k TUA and Top-k PUA, respectively, and show that our framework achieved good accuracy. When ICMP rules were ignored, unexplained sequences were sequences where ICMP activities were occurring, and likewise when preprocessor rules were ignored.

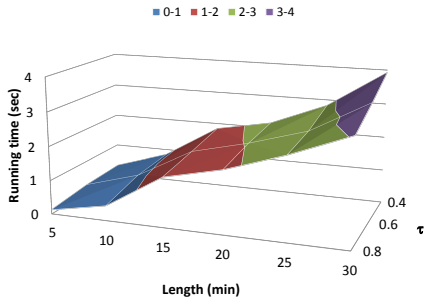
5.3 Experimental Conclusions

Our experiments show that:

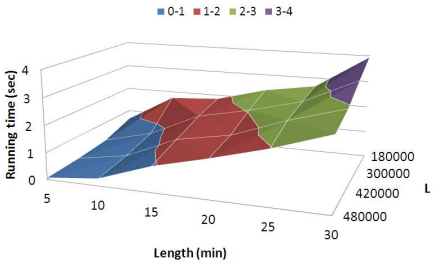
(i) *Runtime increases with observation sequence length* (because there are more possible



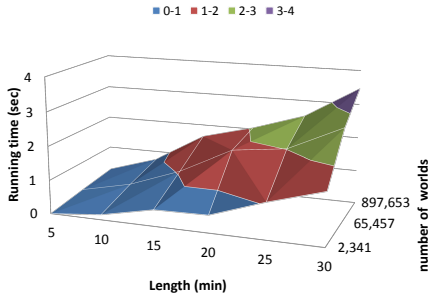
((a)) Varying k ($\tau = 0.6, L = 300000$)



((b)) Varying τ ($k = All, L = 300000$)



((c)) Varying L ($\tau = 0.6, k = All$)



((d)) Varying number of worlds ($\tau = 0.6, k = All, L = 300000$)

Figure 5.20: Running time of Algorithm Top-k PUA on the cyber security dataset.

((a)) Ignoring ICMP rules

τ	Accuracy
0.4	85.71
0.6	71.42
0.8	68.17

((b)) Ignoring Preprocessor rules

τ	Accuracy
0.4	75.12
0.6	63.84
0.8	59.29

Table 5.6: Accuracy of Top-k TUA (cyber security dataset).

worlds, causing $LC(v)$ and $NLC(v)$ to have more variables and constraints). Despite the enormous blow-up in the number of possible worlds, our algorithms perform very well showing quadratic performance in all three datasets.

(ii) *Runtime increases with the number of totally or partially unexplained activities present in the video.* This is because determining the exact endpoints of each TUA (resp. PUA) is costly. Specifically, determining the exact end frame of a TUA requires computing

((a)) Ignoring ICMP rules

τ	Accuracy
0.4	91.24
0.6	83.39
0.8	72.84

((b)) Ignoring Preprocessor rules

τ	Accuracy
0.4	88.76
0.6	76.24
0.8	74.85

Table 5.7: Accuracy of Top-k PUA (cyber security dataset).

$\mathcal{P}_T$ many times: when a TUA is found, Top-k TUA (and also Top-k TUC) need to go through the **while** loop of lines 7–14, the binary search in the **while** loop of lines 16–25, and the **if** block of lines 26–31. All these code blocks require $\mathcal{P}_T$ to be computed. Likewise, determining the exact start and end frames of a PUA requires $\mathcal{P}_P$ to be computed many times as Algorithm Top-k PUA (as well as Algorithm Top-k PUC) goes through different loops and binary searches (one to determine the start frame, another to determine the end frame) requiring multiple computations of $\mathcal{P}_P$.

(iii) *In general, the number of TUAs and PUAs in the observation sequence decreases as τ and L increase*, because higher values of τ and L are stricter conditions for a sequence to be totally or partially unexplained.

(iv) *Runtime decreases as k decreases* because our algorithms use k intelligently to infer that certain sequences are not going to be in the result (aborting the loops and binary searches mentioned above).

(v) *Precision increases whereas recall decreases as τ increases*. The experimental results have shown that a good compromise can be achieved by setting τ at least 0.6 and that our framework had a good accuracy with all the datasets we considered.

Chapter 6

Conclusions

Suppose we have a sequence v of time-stamped observation data and a set $\mathcal{A}$ of "known" activities (normal or suspicious). This thesis addresses the problem of finding subsequences of v that are not "sufficiently" explained by the activities in $\mathcal{A}$. We formally define what it means for a sequence to be unexplained by defining *totally* and *partially* unexplained sequences. We propose a possible worlds framework and identify interesting properties that can be leveraged to make the search for unexplained activities highly efficient via intelligent pruning. We leverage these properties to develop the Top-k TUA, Top-k PUA, Top-k TUC, Top-k PUC algorithms to find totally and partially unexplained activities with highest probabilities. We conducted experimentals over three datasets in the video and cyber security domains showing that our approach has good running time and high accuracy.

This thesis represents a start towards detecting unexplained activities in a domain independent way. Much future work is possible. For instance, we may wish to allow activity occurrences to violate the temporal constraints in the stochastic automata based activity model by penalizing such activity occurrences via diminished probabilities. This can be done in many ways.

Second, we would like to increase scalability of our algorithms, possibly through the development of specialized data structures to support identification of the Top-k TUA, Top-k PUA, Top-k TUC, Top-k PUC algorithms.

Third, the assumption of independence after conflict-based partitioning is convenient and follows upon much work in computer science that makes such assumptions—but it

may not be appropriate for all applications. Relaxing this assumption is also an important direction for future work.

Bibliography

[1] M. Albanese, V. Moscato, A. Picariello, V. S. Subrahmanian, and O. Udrea, "Detecting stochastically scheduled activities in video," in *IJCAI*, pp. 1802–1807, 2007.

[2] S. Hongeng and R. Nevatia, "Multi-agent event recognition," in *ICCV*, pp. 84–93, 2001.

[3] H. Buxton and S. Gong, "Visual surveillance in a dynamic and uncertain world," *Artif. Intell.*, vol. 78, no. 1-2, pp. 431–459, 1995.

[4] T. Huang, D. Koller, J. Malik, G. Ogasawara, B. Rao, S. Russell, and J. Weber, "Automatic symbolic traffic scene analysis using belief networks," in *PROCEEDINGS 12TH NATIONAL CONFERENCE IN AI*, pp. 966–972, AAAI Press, 1994.

[5] S. Hongeng and R. Nevatia, "Multi-agent event recognition," in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 2, pp. 84–91 vol.2, 2001.

[6] S. S. Intille and A. F. Bobick, "A framework for recognizing multi-agent action from visual evidence," in *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, AAAI '99/IAAI '99, (Menlo Park, CA, USA), pp. 518–525, American Association for Artificial Intelligence, 1999.

[7] T. Starner, J. Weaver, and A. Pentland, "Real-time american sign language recognition using desk and wearable computer based video," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 12, p. 1371–1375, 1998.

[8] B. H. Juang and L. R. Rabiner, "Hidden Markov Models for Speech Recognition," *Technometrics*, vol. 33, no. 3, 1991.

[9] A. Kaltenmeier, T. Caesar, J. M. Gloger, and E. Mandler, "Sophisticated topology of hidden Markov models for cursive script recognition," pp. 139–142, 1993.

[10] S. Hongeng, R. Nevatia, and F. Bremond, "Video-based event recognition: activity representation and probabilistic recognition methods," *Comput. Vis. Image Underst.*, vol. 96, pp. 129–162, Nov. 2004.

[11] D. Zhang, D. Gatica-Perez, S. Bengio, and I. McCowan, "Semi-supervised adapted hmms for unusual event detection," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, (Washington, DC, USA), pp. 611–618, IEEE Computer Society, 2005.

[12] M. Brand, N. Oliver, and A. Pentland, "Coupled hidden markov models for complex action recognition," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pp. 994–999, 1997.

[13] N. Vaswani, A. K. R. Chowdhury, and R. Chellappa, ""shape activity": A continuous-state hmm for moving/deforming shapes with application to abnormal activity detection," *IEEE Transactions on Image Processing*, vol. 14, no. 10, pp. 1603–1616, 2005.

[14] M. Brand, N. Oliver, and A. Pentland, "Coupled hidden markov models for complex action recognition," in *CVPR*, pp. 994–999, 1997.

[15] N. Oliver, E. Horvitz, and A. Garg, "Layered representations for human activity recognition," in *ICMI*, pp. 3–8, 2002.

[16] R. Hamid, Y. Huang, and I. Essa, "Argmode - activity recognition using graphical models," in *CVPRW*, pp. 38–43, 2003.

[17] N. P. Cuntoor, B. Yegnanarayana, and R. Chellappa, "Activity modeling using event probability sequences," *IEEE Transactions on Image Processing*, vol. 17, no. 4, pp. 594–607, 2008.

[18] D. Zhang, D. Gatica-Perez, S. Bengio, and I. McCowan, "Semi-supervised adapted hmms for unusual event detection," in *CVPR*, pp. 611–618, 2005.

[19] M. T. Chan, A. Hoogs, J. Schmiederer, and M. Petersen, "Detecting rare events in video using semantic primitives with hmm," in *ICPR*, pp. 150–154, 2004.

[20] T. Xiang and S. Gong, "Video behavior profiling for anomaly detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 5, pp. 893–908, 2008.

[21] J. Kim and K. Grauman, "Observe locally, infer globally: A space-time mrf for detecting abnormal activities with incremental updates," in *CVPR*, 2009.

[22] J. Yin, Q. Yang, and J. J. Pan, "Sensor-based abnormal human-activity detection," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 8, pp. 1082–1090, 2008.

[23] D. H. Hu, X.-X. Zhang, J. Yin, V. W. Zheng, and Q. Yang, "Abnormal activity recognition based on hdp-hmm models," in *IJCAI*, pp. 1715–1720, 2009.

[24] X.-X. Zhang, H. Liu, Y. Gao, and D. H. Hu, "Detecting abnormal events via hierarchical dirichlet processes," in *PAKDD*, pp. 278–289, 2009.

[25] D. Mahajan, N. Kwatra, S. Jain, P. Kalra, and S. Banerjee, "A framework for activity recognition and detection of unusual activities," in *ICVGIP*, 2004.

[26] F. Jiang, Y. Wu, and A. K. Katsaggelos, "Detecting contextual anomalies of crowd motion in surveillance video," in *ICIP*, pp. 1117–1120, 2009.

[27] M. Breunig, H. Kriegel, R. Ng, and J. Sander, "Identifying density-based local outliers," in *Proc. ACM SIGMOD Intl Conf. Management of Data (SIGMOD 00)*, pp. 93–104, 2000.

[28] H. Zhong, J. Shi, and M. Visontai, "Detecting unusual activity in video," in *CVPR*, pp. 819–826, 2004.

[29] C. E. Au, S. Skaff, and J. J. Clark, "Anomaly detection for video surveillance applications," in *ICPR*, pp. 888–891, 2006.

[30] Y. Zhou, S. Yan, and T. S. Huang, "Detecting anomaly in videos from trajectory similarity analysis," in *ICME*, pp. 1087–1090, 2007.

[31] A. Mecocci and M. Pannozzo, "A completely autonomous system that learns anomalous movements in advanced videosurveillance applications," in *ICIP*, pp. 586–589, 2005.

[32] L. Brun, A. Saggese, and M. Vento, "A clustering algorithm of trajectories for behaviour understanding based on string kernels," in *SITIS'12*, pp. 267–274, 2012.

[33] J. Wang, Z. Cheng, M. Zhang, Y. Zhou, and L. Jing, "Design of a situation-aware system for abnormal activity detection of elderly people," in *Active Media Technology* (R. Huang, A. Ghorbani, G. Pasi, T. Yamaguchi, N. Yen, and B. Jin, eds.), vol. 7669 of *Lecture Notes in Computer Science*, pp. 561–571, Springer Berlin Heidelberg, 2012.

[34] B. T. Morris and M. M. Trivedi, "Trajectory learning for activity understanding: Unsupervised, multilevel, and long-term adaptive approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 11, pp. 2287–2301, 2011.

[35] A. Adam, E. Rivlin, I. Shimshoni, and D. Reinitz, "Robust real-time unusual event detection using multiple fixed-location monitors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 3, pp. 555–560, 2008.

[36] F. Jiang, J. Yuan, S. A. Tsaftaris, and A. K. Katsaggelos, "Video anomaly detection in spatiotemporal context," in *ICIP*, pp. 705–708, 2010.

[37] A. Gupta, P. Srinivasan, J. Shi, and L. S. Davis, "Understanding videos, constructing plots learning a visually grounded storyline model from annotated videos," in *CVPR*, pp. 2012–2019, 2009.

[38] M. Albanese, C. Molinaro, F. Persia, A. Picariello, and V. S. Subrahmanian, "Finding unexplained activities in video," in *IJCAI*, pp. 1628–1634, 2011.

[39] P. Ning, Y. Cui, and D. S. Reeves, "Constructing attack scenarios through correlation of intrusion alerts," in *CCS 2002*, (Washington, DC, USA), pp. 245–254, ACM, November 2002.

[40] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers & Security*, vol. 28, pp. 18–28, February-March 2009.

[41] A. Jones and S. Li, "Temporal signatures for intrusion detection," in *ACSAC*, (New Orleans, LA, USA), pp. 252–261, IEEE Computer Society, December 2001.

[42] L. Wang, A. Liu, and S. Jajodia, "Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts," *Computer Communications*, vol. 29, pp. 2917–2933, September 2006.

[43] S. Noel, E. Robertson, and S. Jajodia, "Correlating intrusion events and building attack scenarios through attack graph distances," in *ACSAC*, (Tucson, AZ, USA), pp. 350–359, December 2004.

[44] H. Debar and A. Wespi, "Aggregation and correlation of intrusion-detection alerts," in *RAID* (W. Lee, L. Mé, and A. Wespi, eds.), vol. 2212 of *Lecture Notes in Computer Science*, (Davis, CA, USA), pp. 85–103, Springer, October 2001.

[45] S. O. Al-Mamory and H. Zhang, "Ids alerts correlation using grammar-based approach," *Journal of Computer Virology*, vol. 5, pp. 271–282, November 2009.

[46] J. P. Anderson, "Computer security threat monitoring and surveillance," tech. rep., James P. Anderson Co., Fort Washington, PA, USA, April 1980.

[47] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network intrusion detection," *IEEE Network*, vol. 8, pp. 26–41, May 1994.

[48] X. Qin and W. Lee, "Statistical causality analysis of INFOSEC alert data," in *RAID* (G. Vigna, C. Kruegel, and E. Jonsson, eds.), vol. 2820 of *Lecture Notes in Computer Science*, (Pittsburgh, PA, USA), pp. 73–93, Springer, September 2003.

[49] X. Qin, *A Probabilistic-Based Framework for INFOSEC Alert Correlation*. Phd thesis, Georgia Institute of Technology, August 2005.

[50] A. J. Oliner, A. V. Kulkarni, and A. Aiken, "Community epidemic detection using time-correlated anomalies," in *RAID* (S. Jha, R. Sommer, and C. Kreibich, eds.), vol. 6307 of *Lecture Notes in Computer Science*, (Ottawa, Canada), pp. 360–381, Springer, September 2010.

[51] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa, "A new algorithm for generating all the maximal independent sets," *SIAM J. Comput.*, vol. 6, no. 3, pp. 505–517, 1977.

[52] N. T. Siebel and S. Maybank, "Fusion of multiple tracking algorithms for robust people tracking," in *In Proc. of ECCV02*, pp. 373–387, 2002.

[53] N. T. Siebel and S. J. Maybank, "The advisor visual surveillance system," in *in ECCV 2004 workshop Applications of Computer Vision (ACV*, 2004.

[54] M. Albanese, A. Pugliese, and V. Subrahmanian, "Fast activity detection: Indexing for temporal stochastic automaton-based activity models," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 2, pp. 360–373, 2013.

[55] "Ninth IEEE international workshop on performance evaluation of tracking and surveillance (PETS 2006) benchmark data." `http://www.cvg.rdg.ac.uk/PETS2006/data.html`, 2006.